2006

Semantic tableaux program

Sirisha Lakshmi Vadaparty

SEMANTIC TABLEAUX PROGRAM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the degree

Master of Science

in

Computer Science

---

by

Sirisha Lakshmi Vadaparty

March 2006

SEMANTIC TABLEAUX PROGRAM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

by

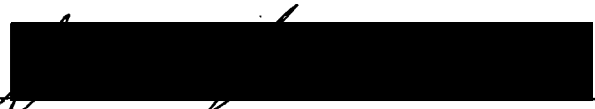
Sirisha Lakshmi Vadaparty

March 2006

Approved by:

Dr. Richard Botting, Chair, Computer Science — Date

Dr. Kerstin Voigt



Dr. George Georgiou

ABSTRACT

Formal Methods is a specialized field in Computer Science that uses mathematical logic to specify, design and verify a hardware or software system. By modeling situations by way of Formal Methods a computer scientist can reason about the situation. Although some question the value of, Formal Methods, it is important in safety critical systems. For example, NASA has a project devoted to the study of Formal Methods in it's systems [1].

Semantic Tableaux are based on truth trees and has been considered the most popular deduction style proving technique especially among students [R.Botting (personal communication, September 20, 2005)].

This project created a program that takes predicate calculus formulas and creates a visual Semantic Tableaux truth tree, thereby proving or disproving a conclusion. This project has been implemented in Java and will be available over the web. This project should not only promote the subject of Semantic Tableaux but be useful help for computer scientist in modeling situations for their computer programs. There will be hope in the coming years that this project will be improved upon.

ACKNOWLEDGMENTS

I would like to give thanks to the entire Computer Science Department at California State University San Bernardino for their support in helping me towards my goal of obtaining a MS degree. In particular I would like to thank Dr. Richard Botting, Dr. Kerstin Voigt, and Dr. George Georgiou.

I would also like to thank the support of the National Science Foundation under the award 9810708.

I like to dedicate this work to my family, especially to my mother.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

SOFTWARE REQUIREMENT SPECIFICATION

1.1 Introduction

The object of this Computer Science project is the creation of a Java program for a web site. The project is based on the subject of Formal Methods, in particular, the subject of Semantic Tableaux [2]. Semantic Tableaux is an easy to understand proof method. Such methods are also called truth trees [3]. The program is interactive and will allow the user to input predicate formulas as assumptions and a conclusion. The program then creates and displays a visual Semantic Tableaux tree. If each branch of this tree contains a contradiction then the argument is valid.

For example, suppose we have two assumptions such as "All dogs bark" and "Jane's pet barks" and a conclusion "Therefore, Jane's pet is a dog". This is an invalid argument because some barking animals are not dogs. In this case the program will give the user a counter example. An example of a correct argument would be the two assumptions "If a pet barks it is a dog" and "Jane's pet barks", plus a conclusion of "Jane's pet is a dog". The program allows the user to input the two assumptions and conclusion as well-

formed formulas. Without loss of generality, only two assumptions are needed. Then the program shows the user the Semantic Tableaux as a deduction style proof [4]. If the argument is valid, the assumptions implies the condition. The proof method is a refutation method that derives contradictions from assuming the assumptions plus the negation of the conclusion. Propositional Calculus and Lower Predicate Calculus will both be used by the program.

Semantic Tableaux method is a proof using contradiction and case distinction method [5]. See APPENDIX B for Semantic Tableaux rules by W.Hodges. The illustrations of the rules of Semantic Tableaux were taken with permission from Dr.Richard Botting's web site. The website containing the rules of Semantic Tableaux is from www.csci.csusb.edu/dick/math/semtab.gif (see APPENDIX B). The website containing examples of Semantic Tableaux is from www.csci.csusb.edu/dick/maths/semtaex.jpg (see APPENDIX C).

Truth trees are useful because they eliminate a whole group of cases at once [6].This method grows a tree containing branches. These branches can be open or closed. A closed branch has the two contradictory formulas appearing in it. Otherwise it is open. In a closed tree all

the branches are closed and this proves the validity of the argument or sequent. The following table has the notations defined.

Table 1. Notations

Notation	Example	Definition
!	P!	Not P
¬	¬P	Not P
∨	P ∨ Q	P or Q
∧	P ∧ Q	P and Q
→	P→Q	If P then Q
↔	P↔Q	P if and only if Q
∀	∀(x)(P(x)→Q(x))	For all x, if P(x) then Q(x)
∃	∃(x)(P(x)∧Q(x)	There exits x, P(x) and Q(x)

The following illustration gives an example of a closed tree. If the end of the branch has an "x" that means the leaf/end has two formulae that are contradictory. Contradictory formulae such as P and ¬P closes the left

branch, and Q and ¬Q are contradictory formulae that closes the right branch. Contradictory formulae will close the branch regardless of how far apart they are. The "x" can also indicate the derivation of the formula t!= t for some term or expression t, which contradicts one of the equality axioms.

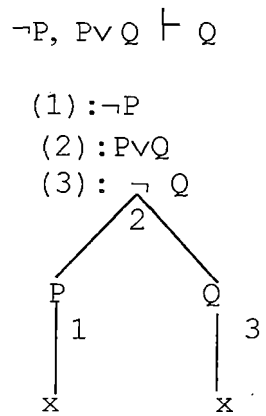


Figure 1. Closed Tree

Semantic Tableaux method "allows one to systematically generate subcases until elementary contradictions are reached"[7]. When a formula is can be checked off, you no longer have to do anything to it (see APPENDIX B). If it is not checked off, it means more substitutions may be required.

Following are two examples of a closed tableaux.

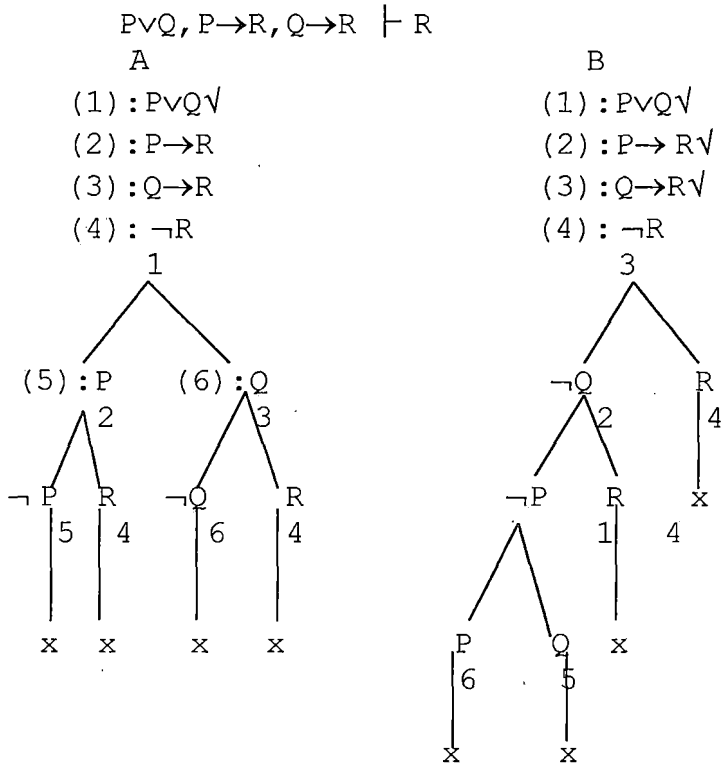


Figure 2. Closed Tableaux

(4): ¬R is proved by counterexample. (1) is selected first in 2A. According to the rules of Semantic Tableaux rule by W.Hodges (see APPENDIX B), P∨Q expands to P and Q in separate branches. The check mark is added at step one to show that it will not be needed again. Then the user selects (2) and (3). (2) is P→R (if P then R) and according the rule by W.Hodges is should expand into two branches of ¬P and R . (3) is Q→R and this should follow the same rule as (2). When P meets ¬P, Q meets ¬Q, and R meets ¬R the Semantic Tableaux tree is proved by

contradiction. In 2B the user selects (3) then (2),then (1). Although the order is different, by following the same rules set by W.Hodges the tableaux is proven by contradiction as well.

Sometimes according to the rule by W. Hodges it is necessary to substitute a variable or expression for a variable that is bound in a formula. In this case it is important to define a bound variable and a free variable. A bound variable is within the scope of a quantifier and a free variable isn't. A variable that has no quantifier in a formula is a free variable. The identifier "x" in the formula "$(P(x)\vee Q(x))\rightarrow \forall(x)(P(x)\rightarrow Q(x))$" has an example of a bound and free variable. "x" is a bound variable in the scope of a quantifier "$\forall$". Following is an illustration of a parse tree of the above formula. This parse tree identifies the scope of the free and bound variable.

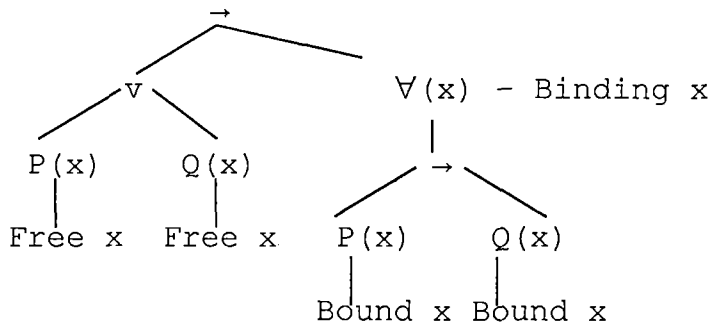


Figure 3. Free and Bound Variables

In some cases a capture of a variable occurs when a free variable intersects with a bound variable in an expression. This can be best explained in a passage from "Logic in Computer Science" a book by Michael R.A. Huth and Mark D. Ryan[8].

> Unfortunately, substitutions can give rise to undesired side effects. In performing a substitution $\Phi[t/x]$, the term t may contain a variable y, where free occurrences of x in Φ are under the scope of $\exists y$ or $\forall y$ in Φ. By carrying out this substitution $\Phi[t/x]$, the values y, which might have been fixed by a concrete context, get caught in the scope of $\exists y$ or $\forall y$. This binding capture overrides the context specification of the concrete value of y, for it will now stand for 'some unspecified' or 'all', respectively.

Such undesired variable captures are to be avoided at all costs.

The solution above is to change the 'y' to another identifier first.

The program can be used to explore the implications of a set of formulas. Semantic Tableaux always derives a set of alternative sound conclusions. In the Propositional Calculus each path is associated with a conjunct in the Disjunctive Normal Form. So a single open branch can be used to establish a valid sequent. This is useful when the user wants to explore the consequences of assuming some WFFs but doesn't have a particular conclusion in mind. The following illustration is an example. Here the only open branch contains Q, so we can conclude that if P then P→Q then Q must be true. This is the well known rule of "modus ponens"[9].

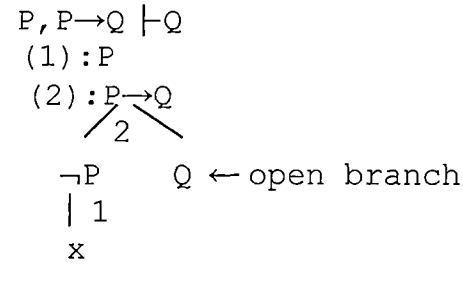


Figure 4. Open Branch

1.2 Purpose of this Project

The purpose of this project is to create an interactive program that will create a Semantic Tableaux Tree after entering at least one WFF. This program aids the user in the creation of a Semantic Tableaux tree that might be harder and more tedious by hand. It provides real help for those in the study of Formal Methods. It could also be a fun teaching tool in learning about Discrete Mathematics.

At the start of this project there was no known program that will allow a user to create a Semantic Tableaux Tree. Since then Bertie and Twootie [10] are two such predicate logic programs available. Both are written in Pascal. Bertie is a proof checker for natural deduction systems in sentential and predicate logic. Twootie is a tool for exploring truth trees in predicate and sentential logic. Twootietwo is a recent program that is based on Java applet like mine [11]. This program was found recently and developed independently of this project. However there seems to be no program that will create a visual Semantic Tableaux tree and to use this tree as a proof. Also this program is far more interactive allowing the user to direct how he or she wants the truth tree to be arranged.

There was a Master's Project done by Jimmy Lee for the

CSUSB [12]. His project was about And/Or tables. There was hope of building on his project but after looking at his work, it was decided to build this project from scratch. This program will provide a practical tool for predicate calculus.

The program is for anybody having a sound understanding of Formal Methods. It should be useful for beginners and advance students of Formal Methods. The user, in order to use the program, will need access to a computer that has access to web. The user's browser must have a Java Virtual Machine. As long the program is running on the server, user should be able to access it. This project can help people learn about Semantic Tableaux, promote the subject of Formal Methods and its importance to the subject Computer Science.

This program requires user input on occasion for assistance in building the Semantic Tableaux tree. The program as of yet can not allow the user to undo steps. That is, to erase the tree to a certain point and rebuild the tree without having to start all over. The program can not yet save a work in progress to be worked on later.

1.3 Definitions

Table 2. Terms and Definitions

Apache	A program that operates a web server
CSCI	Computer Science Department at California State University San Bernardino
CSUSB	California State University San Bernardino
GUI	Graphic User Interface
HTML	Hypertext Markup Language
Java	A programming language developed by Sun Mircosystems
Operator	A function that performs a task on Predicates.
Predicate	Defines or describes a property of an object.
Quantifier	An operator that limits the variables in a first

	order predicate calculus.
SDK	Software Development Kit
Tomcat	A servlet container used for implementation of Java servlet and Server Pages Technologies
Variable	A symbolic identifier for a program state or a unknown value.
WFF	Well-Formed Formula

1.4 Preliminary Design

The program has three HTML pages. The Main page is the introduction page. The Input page is where the user inputs the required predicate calculus. And there is a Semantic Tableaux page. This page builds the Semantic Tableaux tree. There will be no password associated with this site. Anyone can use it. See APPENDIX D for the Use Case diagram.

1.4.1 Main Page

This page is an introduction page. It introduces the project, gives some basic instructions, and explains the program capabilities.

1.4.2 Input Page

Here the user will input the well-formed formula (WFF). The user will have the option to input one formula or three. Then a parser program converts the input to a data structure or if there was any errors the program will terminate. Once the formulas are correct the program will go to the Semantic Tableaux page.

1.4.3 Semantic Tableaux Page

Here the user sees the program create the Semantic Tableaux tree. If the predicate calculus formulas are complicated, the program requires user input. The program will have the appropriate GUI interface. If there is any

problem the program will terminate. Such a problem is that the predicate calculus given is incorrect and won't create a proper binary tree. The user could decide to terminate the program and start over. If there are no problems the user should see the proper Semantic Tableaux tree proving or disproving the logic.

CHAPTER TWO

SYSTEM DESIGN

2.1 Deployment Diagram

The diagram below shows the UML 1.5 deployment diagram.

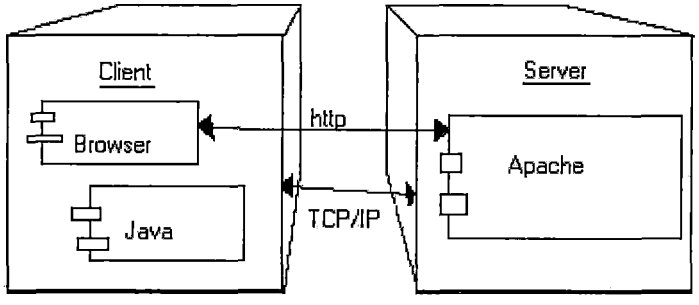


Figure 5. Deployment Diagram(UML 1.5)

2.2 System Components

Java is an appropriate computer language for the web. It was chosen because it is an Object Oriented Program that would allow modification of the program at a later date. It also has GUI interface which is necessary for the project.

2.3 Software Interfaces

Internet Brower: No known browser constraints

Client Operating System (OS): No known constraints. Tested on Windows XP. Unicode may not show up on all browsers. Tested on Internet Explorer Version 6.0.2900

Server Operating System (OS): RedHat Linux

Web Server: Apache

Servlet: Jakarta Tomcat

CHAPTER THREE

IMPLEMENTATION

3.1 Introduction

The main goal of the program is to create a Semantic Tableaux Tree with no or little assistance. The program was designed to be as user friendly as possible.

3.2 Input Page

The Input page is where the user will input the required predicate calculus. It is a Graphic User Interface (GUI) by means of a Java applet using AWT classes. A parsing program takes the formula and parses it into a vector. The parsing program won't just take any string and parse it into a vector. The formula must be in a certain format or an error will occur. Once the formula is parsed into a vector, another program will then create a binary tree out of the vector. The creation of the binary tree depends on three programs BinaryTree.java, ChildNode.java, and BuildTree.java. The first two programs are based on pGenericBinaryTree.java and PTwoChildNode.java programs. These programs are from the website called Javacommerce [13]. These two programs were used with permission. If the program cannot create a binary tree out of the vector that

means the predicate formulas was incorrect despite the fact it was parsed with no problems. See the following figure to see how the GUI input applet looks like.

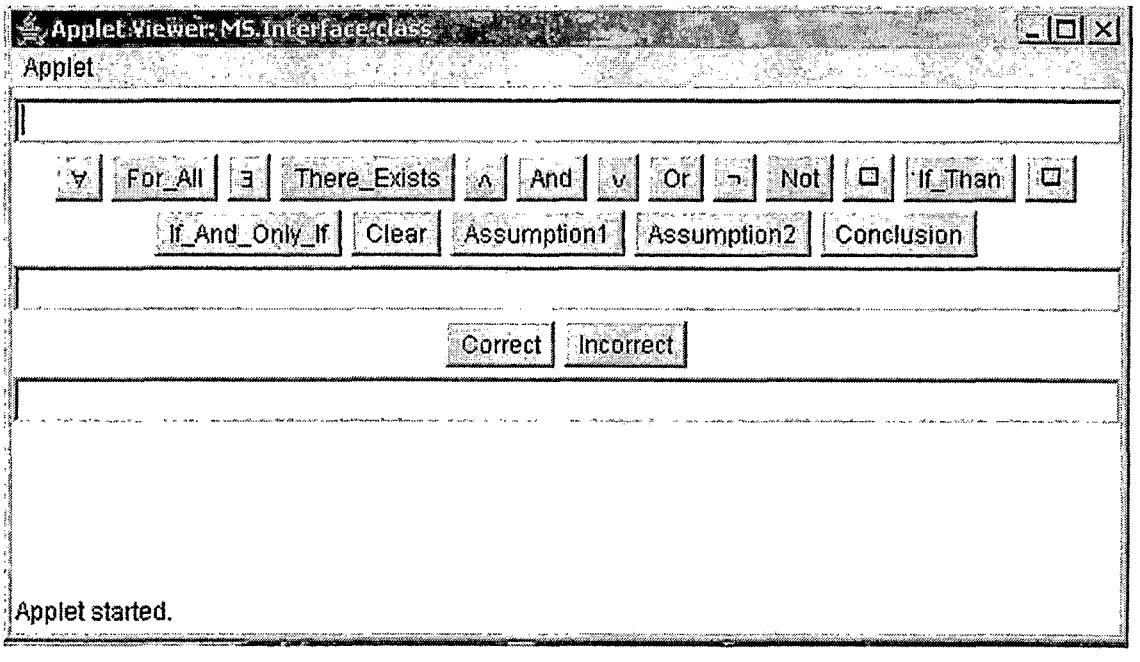
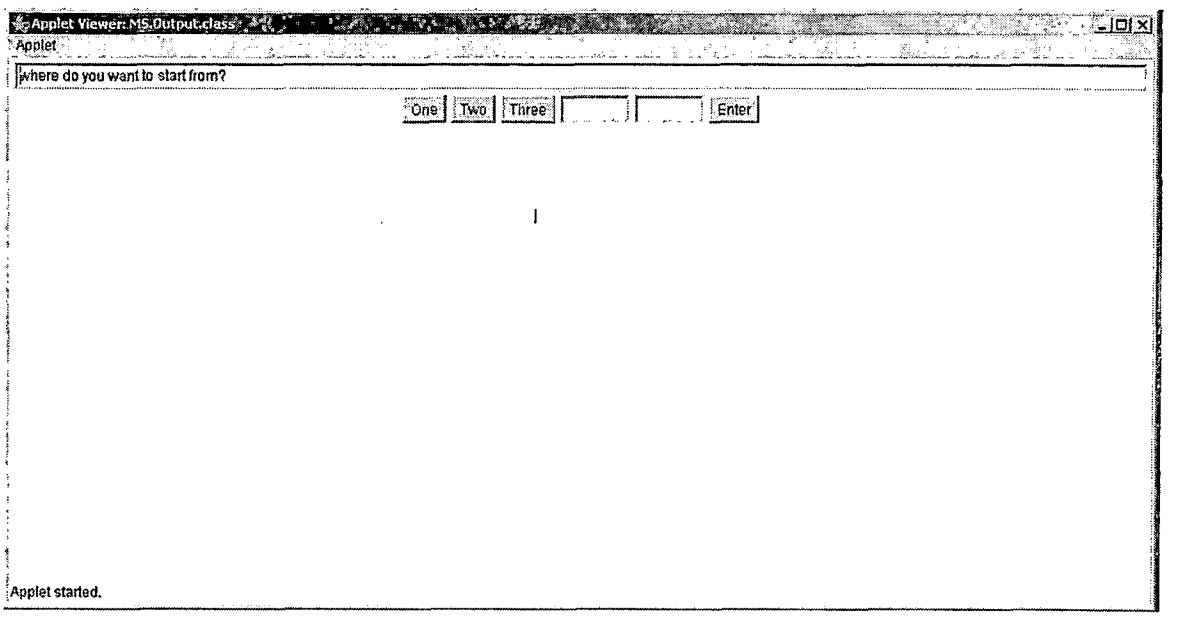


Figure 6. Input Applet

3.3 Semantic Tableaux Tree

Here is where the program starts to build the Semantic Tableaux tree. The information passed from Input applet to Output applet is done by using a third Java program. The idea was gotten from the web site Java World [14]. This is also a GUI applet. If the formula is simple, the program creates tree by itself. The program takes the binary tree that was created before and look at the appropriate node in

that tree. And in that node should be a quantifier, predicate, operator or a variable. With this information, a new tree is created. This tree is the Semantic Tableaux Tree. This Semantic Tableaux tree finished or not will be seen by the user. That way the user should see the progress of the tree. See the following figure to illustrates how the GUI applet output looks like.



Fiqure 7. Output Applet

CHAPTER FOUR

VERIFICATION AND VALIDATION

To insure that the program is dependable and works as expected a set of test was performed. The tests see if the program works correctly and consistently.

4.1 Unit Test

Each part of the program was tested individually to insure that the program works by itself and to insure that it will work in an over all Object Oriented Design. Each Java file was tested. See APPENDIX E for results.

4.2 Subsystem Testing

In this testing certain related Java files were tested to see if they worked together. This was to fix any noticeable problems that would be harder to detect if all the programs were in use together. For example, parsing program was tested with the binary tree program to see if they work properly together. See APPENDIX F for results.

4.3 Project Testing

The final project itself was tested to see if it operates as intended. See APPENDIX G for results.

CHAPTER FIVE

MAINTENANCE MANUAL

The information here will provide the instructions that will be used to setup the system.

5.1 Software Installation

The requirements for this system are RedHat Linux as the operating system and Apache as the web server. The servlet that is used is Tomcat. Java is an Object Oriented Program that is particular useful in the creation of programs to run within a web browser.

5.1.1 RedHat Installation

RedHat is a popular operation system. This project used RedHat version 9 that was available for CSUSB students. To use RedHat the following instructions must be followed.

Install CD 1 into the CD-ROM. Then, start up the machine.

You should see the install wizard. Set up the system according to your needs such as network settings.

5.1.2 Apache Installation

When installing Apache make sure you request Apache in the install wizard of RedHat.

5.1.3 Java Software Development Kit Installation

Download Sun's Java 2 Platform, Standard Edition. One can be downloaded from http://java.sun.con/j2se/. The following directions are from http://www.linuxgazette.com/issue95/millson.html.

Make a self-extraction binary executable in the directory where you downloaded the Software Development Kit. Run the two following commands.

```
Chmod +x j2sdk-1_4_2-ia586.bin
./j2sdk-1_4_2-linux-i586.bin
```

A directory called j2sdk1.4.2 should exist in the download directory. Create a "/usr/java". Move the SDK directory to "/usr/java" using the following command.

```
mv j2sdk1.4.2 /usr/java
```

Now set the environment variable called JAVA_HOME by altering the /etc/profile by adding the following two lines.

```
Java_Home="usr/java/j2sdk1.4.2"
Export Java_Home.
```

5.1.4 Tomcat Installation

The following commands will create a group and user account for Tomcat. This will also create a "/home/tomcat" directory.

Groupadd tomcat

Useradd -g tomcat tomcat

Tomcat latest build can be downloaded from http://www.apache.org/dist/jakarta/tomcat-4/binaries/.

tar xvzf tomcat-4.1.27-LE-jek14.tar.gz

The previous command will unzip Tomcat.

mv Jakarta-Tomcat-4.1.27-LE-jdk14 /usr/local/

This command will move the file to the directory where Tomcat will be installed.

CATALINA_HOME is what the directory where Tomcat is installed is referred to.

Setup a symbolic link by using the following command.

ln -s Jakarta-tomact-4.1.27-LE-jdk14 jakarta-tomcat

For "/usr/local/Jakarta-tomcat" and "/usr/local/Jakarta-tomcat-4.1.27-LE-jdk14", the group and owner need to be changed. Do so by using the following two commands.

chown tomcat.tomcat /usr/local/Jakarta-tomcat

chown -R tomcat.tomcat /usr/local/Jakarta-tomcat-4.1.27-LE-jdk14 .

CHAPTER SIX

CONCLUSION AND FUTURE ENCHANCEMENTS

6.1 Conclusion

The important need of using truth trees as proof checkers is shown by the increasing number of programs created for this task. This project created an important program that was not previous available. A Java based interactive truth tree program that will allow the user to direct the direction of the truth tree. The system was designed to be as user friendly as possible. This is a highly effective tool in the study of Formal Method and Semantic Tableaux in particular. The system can accept further enhancements. One area is that no matter how complicated the predicate calculus formula is the program should give the option to create a Semantic Tableaux tree automatically. It should also give the option to go back or reset to a certain point the creation of the tree without having to restart all over again. And it should allow the user to save a work in progress so the user can work it on a later date.

APPENDIX A

SOURCE CODE

```
/*************************************************************
Name:   ChildNode.java
Description:   Data Structure for a Child Node.
Data: January 2006
*************************************************************/

package MS;

import java.lang.Object;
import java.io.*;

public class ChildNode{
    protected String LData,RData, data, predicate, quantifier, operator, variable1,
    variable2, leftTree, rightTree, nodeTree;
    protected String DTestClose, RTestClose, LTestClose;
    protected ChildNode parent,left,right;
    protected boolean not,Lnot,Rnot, center,checkedOff, RcheckedOff,LcheckedOff, closed,
    Rclosed, Lclosed, lineClosed;
    protected boolean clicked, Rclicked, Lclicked;
    protected BinaryTree dataTree, rightDataTree, leftDataTree ;

    public ChildNode(){
        data=LData=RData = null;
        parent=left = right = null;

        quantifier = predicate = variable1 = variable2 = leftTree = rightTree = nodeTree
        = null ;
        DTestClose=RTestClose=LTestClose=null;
        not = false ; Lnot = false; Rnot = false; center =
        false;checkedOff=false;closed=false;
        RcheckedOff=false; LcheckedOff=false; Rclosed=false; Lclosed=false;
        clicked=false; Rclicked=false; Lclicked=false;
        dataTree=rightDataTree=leftDataTree=null;
    }
    public ChildNode(String d){
        data=LData=RData = d;
        parent=left = right = null;
        DTestClose=RTestClose=LTestClose=null;
        quantifier = predicate = variable1 =  variable2 = leftTree = rightTree = nodeTree
        =null  ;
        dataTree=rightDataTree=leftDataTree=null;
        DTestClose=RTestClose=LTestClose=null;
    }
    public void setLeft(ChildNode l){
        left = l;
    }
    public void setRight(ChildNode r){
        right = r;
    }
    public void setParent(ChildNode p){
        parent = p;
    }
    public void setData(String d){
        data = d;
    }
    public void setLeftData(String d){
        LData = d;
    }
    public void setRightData(String d){
        RData = d;
    }
    public void setQuantifier(String d){
        quantifier = d ;
    }
    public void setPredicate(String d){
        predicate = d ;
```

```
}
public void setOperator(String d){
    operator = d ;
}
public void setVariable1(String d){
    variable1 = d ;
}
public void setVariable2(String d){
    variable2 = d ;
}
public void setDTestClose(String d){
    DTestClose = d;
}
public void setRTestClose(String d){
    RTestClose = d;
}
public void setLTestClose(String d){
    LTestClose = d;
}
public void setDataTree(BinaryTree d){
    dataTree=d;
}
public void setRightDataTree(BinaryTree d){
    rightDataTree=d;
}
public void setLeftDataTree(BinaryTree d){
    leftDataTree=d;
}
public void setLeftTree(String d){
    leftTree=d;
}
public void setRightTree(String d){
    rightTree=d;
}
 public void setNodeTree(String d){
    nodeTree=d;
}
public void setNot(){
    not=!not;
}

public void setLNot(){
    Lnot=!Lnot;
}
public void setRNot(){
    Rnot=!Rnot;
 }
public void setCenter(){
    center=!center;
}
public void setCheckedOff(){
    checkedOff=!checkedOff;
}
public void setRCheckedOff(){
    RcheckedOff=!RcheckedOff;
}
public void setLCheckedOff(){
    LcheckedOff=!LcheckedOff;
}
public void setClosed(){
    closed=!closed;
}
 public void setRClosed(){
     Rclosed=!Rclosed;
 }
 public void setLClosed(){
     Lclosed=!Lclosed;
```

```
 }
 public void setLineClosed(){
     lineClosed=!lineClosed ;
 }
public void setClicked(){
     clicked=!clicked();
 }
public void setRClicked(){
     Rclicked =!Rclicked();
 }
public void setLClicked(){
     Lclicked=!Lclicked();
 }
 public ChildNode getLeft(){
     return left;
 }
 public ChildNode getRight(){
     return right;
 }
 public ChildNode getParent(){
     return parent;
 }
 public boolean getNot() {
     return not ;
 }
 public boolean getLnot() {
     return Lnot ;
 }
 public boolean getRnot() {
     return Rnot ;
 }
 public boolean getCenter() {
     return center ;
 }
 public boolean getCheckedOff() {
     return checkedOff ;
 }
 public boolean getRCheckedOff() {
     return RcheckedOff ;
 }
 public boolean getLCheckedOff() {
     return LcheckedOff ;
 }
 public boolean getClosed() {
     return closed ;
 }
 public boolean getRClosed() {
     return Rclosed ;
 }
 public boolean getLClosed() {
     return Lclosed ;
 }
 public boolean getLineClosed() {
     return lineClosed ;
 }
 public boolean getClicked() {
     return clicked;
 }
 public boolean getRClicked() {
     return Rclicked ;
 }
 public boolean getLClicked() {
     return Lclicked ;
 }
 public String getData(){
     return data;
 }
```

```
    public String getLeftData(){
        return LData;
    }
    public String getRightData(){
        return RData;
    }
    public String getDTestClose(){
        return DTestClose;
    }
    public String getRTestClose(){
        return RTestClose;
    }
    public String getLTestClose(){
        return LTestClose;
    }
    public String getQuantifier(){
        return quantifier;
    }
    public String getOperator(){
        return operator;
    }
    public String getPredicate(){
        return predicate;
    }
    public String getVariable1(){
        return variable1;
    }
    public String getVariable2(){
        return variable2;
    }
    public BinaryTree getDataTree(){
         return dataTree;
    }
    public BinaryTree getRightDataTree(){
         return rightDataTree;
    }
    public BinaryTree getLeftDataTree(){
         return leftDataTree;
    }
     public String getLeftTree(){
         return leftTree;
    }
     public String getRightTree(){
         return rightTree;
    }
    public String getNodeTree(){
         return nodeTree;
    }

}
/*************************************************************
Name:   BinaryTree.java
Description:   Data Structure for a Binary Tree.
Data: January 2006
*************************************************************/

package MS;

import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class BinaryTree extends ChildNode{

    private ChildNode root;
```

```
    /*branches 1-5 will be assigned as regular branches. There will only
be five because this is a the first version of this program. but latter on
more can be added. Branches 6-11 will be assigned to special branches such
as formulas that are not checked off according to the rules by W.Hodges.*/

    private ChildNode current,branch1,branch2,branch3,branch4,branch5branch6,
                              branch7,branch8,branch9,branch10,branch11;
    private ChildNode temp;
    private ChildNode newnode;
    private ChildNode newnodeL;
    private ChildNode newnodeR;

    protected void setRoot(ChildNode r){
        root = r;
        current = r; branch1=r;
        branch2=r; branch6=r;
        branch3=r; branch7=r;
        branch4=r; branch8=r;
        branch5=r; Branch9=r;
        branch10=r; branch11=r;
    }
    public void setCurrent(ChildNode n)  {
        current = n ;
    }
    public BinaryTree(){
        setRoot(null);
    }
    public boolean isEmpty(){
        return getRoot() == null;
    }
    public ChildNode getCurrent() {
        return current ;
    }
    public ChildNode getRoot() {
        return root ;
    }
    public ChildNode getBranch1() {
        return branch1 ;
    }
    public ChildNode getBranch2() {
        return branch2 ;
    }
    public ChildNode getBranch3() {
        return branch3 ;
    }
    public ChildNode getBranch4() {
        return branch4 ;
    }
    public ChildNode getBranch5() {
        return branch5 ;
    }
    public ChildNode getBranch6() {
        return branch6 ;
    }
    public ChildNode getBranch7() {
        return branch7 ;
    }
    public ChildNode getBranch8() {
        return branch8 ;
    }
    public ChildNode getBranch9() {
        return branch9 ;
    }
    public ChildNode getBranch10() {
        return branch10 ;
    }
    public ChildNode getBranch11() {
```

```
    return branch11 ;
}
public void setBranch1(ChildNode b) {
    branch1= b ;
}
public void setBranch2(ChildNode b) {
    branch2= b;
}
public void setBranch3(ChildNode b) {
    branch3=b ;
}
public void setBranch4(ChildNode b) {
    branch4=b ;
}
public void setBranch5(ChildNode b) {
    branch5= b ;
}
public void setBranch6(ChildNode b) {
    branch6= b ;
}
public void setBranch7(ChildNode b) {
    branch7= b ;
}
public void setBranch8(ChildNode b) {
    branch8= b ;
}
public void setBranch9(ChildNode b) {
    branch9= b ;
}
public void setBranch10(ChildNode b) {
    branch10= b ;
}
public void setBranch11(ChildNode b) {
    branch11= b ;
}
public boolean getNot() {
    return getCurrent().getNot() ;
}
public boolean getLnot() {
    return getCurrent().getLnot() ;
}
public boolean getRnot() {
    return getCurrent().getRNot() ;
}
public boolean getCenter() {
    return getCurrent().getCenter() ;
}
public boolean getCheckedOff() {
    return getCurrent().getCheckedOff() ;
}
public boolean getClosed() {
    return getCurrent().getClosed() ;
}
public String getData(){
    if(!isEmpty())
        return getCurrent().getData();
    return null;
}
public String getRightData(){
    if(!isEmpty())
        return getCurrent().getRightData();
    return null;
}
public String getLeftData(){
    if(!isEmpty())
        return getCurrent().getLeftData();
    return null;
```

```
}
public String getQuantifier(){
    if(!isEmpty())
        return getCurrent().getQuantifier();
    return null;
}
public String getOperator(){
    if(!isEmpty())
        return getCurrent().getOperator();
    return null;
}
public String getPredicate(){
    if(!isEmpty())
        return getCurrent().getPredicate();
    return null;
}
public String getVariable1(){
    if(!isEmpty())
        return getCurrent().getVariable1();
    return null;
}
public String getVariable2(){
    if(!isEmpty())
        return getCurrent().getVariable2();
    return null;
}
public ChildNode getLeft(){
    if(!isEmpty())
       { current = getCurrent().getLeft() ;
         return getCurrent().getLeft() ;
       }
    else
    return null;
}
public ChildNode getRight(){
    if(!isEmpty())
        { current = getCurrent().getRight();
          return getCurrent().getRight();
        }
    else
    return null;
}
 public BinaryTree getDataTree(){
    if(!isEmpty())
        { current = getCurrent().getDataTree();
          return getCurrent().getDataTree();
        }
    else
    return null;
}
public BinaryTree getRightDataTree(){
    if(!isEmpty())
        { current = getCurrent().getRightDataTree();
          return getCurrent().getRightDataTree();
        }
    else
    return null;
}
public BinaryTree getLeftDataTree(){
    if(!isEmpty())
        { current = getCurrent().getLeftDataTree();
          return getCurrent().getLeftDataTree();
        }
    else
    return null;
}
public String getLeftTree(){
```

```
    if(!isEmpty())
        {
          return getCurrent().getLeftTree();
        }
    else
    return null;
}
public String getRightTree(){
    if(!isEmpty())
        {
          return getCurrent().getRightTree();
        }
    else
    return null;
}
public String getNodeTree(){
    if(!isEmpty())
        {
          return getCurrent().getNodeTree();
        }
    else
    return null;
}
public ChildNode getParent(){

    current = getCurrent().getParent();
    return getCurrent().getParent();
}
public void setNot(){
    current.setNot();
}
public void setLNot(){
    current.setLNot();
}
public void setRNot(){
    current.setRNot();
}
public void setCenter(){
    current.setCenter();
}
public void setCheckedOff(){
    current.setCheckedOff();
}
public void setClosed(){
    current.setClosed();
}
public void inData(String o){
    current.setData(o);
}
public void inLeftData(String o){
    current.setLeftData(o);
}
public void inRightData(String o){
    current.setRightData(o);
}
public void inQuantifier(String o){
    current.setQuantifier(o);
}
public void inOperator(String o){
    current.setOperator(o);
}
public void inPredicate(String o){
    current.setPredicate(o);
}
public void inVariable1(String o){
    current.setVariable1(o);
}
```

```
public void inVariable2(String o){
    current.setVariable2(o);
}
public void inDataTree(BinaryTree o){
    current.setDataTree(o);
}
public void inRightDataTree(BinaryTree o){
    current.setRightDataTree(o);
}
public void inLeftDataTree(BinaryTree o){
    current.setLeftDataTree(o);
}
public void inLeftTree(String o){
    current.setLeftTree(o);
}
public void inRightTree(String o){
    current.setRightTree(o);
}
public void inNodeTree(String o){
    current.setNodeTree(o);
}
public void insertLeftGo(){
    newnode = new ChildNode();
    temp = getCurrent() ;
    temp.setLeft(newnode) ;
    newnode.setParent(temp);
    current = temp.getLeft();

}
public void insertLeft(){
    newnode = new ChildNode();
    temp = getCurrent() ;
    temp.setLeft(newnode) ;
    newnode.setParent(temp);

}
public void insertRightGo(){
    newnode = new ChildNode();
    temp = getCurrent() ;
    temp.setRight(newnode) ;
    newnode.setParent(temp);
    current = temp.getRight();

}
public void insertRight(){
    newnode = new ChildNode();
    temp = getCurrent() ;
    temp.setRight(newnode) ;
    newnode.setParent(temp);

}
public void insertParentGo(){
    newnode = new ChildNode(o);
    temp = getCurrent() ;
    temp.setParent(newnode);
    newnode.setLeft(temp);
    current = temp.getParent();

}
public void insertParent(){
    newnode = new ChildNode();
    temp = getCurrent() ;
    temp.setParent(newnode);
    newnode.setLeft(temp);
}
protected void pretrav(ChildNode p){
    if(p == null)
```

```
            return;
          System.out.println(p.getData()+" =data, " +
           p.getNot()+" =not, " +
           p.getQuantifier()+" =quantifier, " +
           p.getOperator()+" =operator, " +
           p.getPredicate()+" =predicate, " +
           p.getVariable1()+ " =variable1, " +
           p.getVariable2()+ " =variable2, " +
           p.getLnot()+" =Lnot, " +
           p.getLeftData()+ " =left data ," +
           p.getRightData()+ " =Right data ."+
           p.getRnot()+" =RNot, " +
           p.getCenter()+ " =center." );
           pretrav(p.getLeft());
           pretrav(p.getRight());
    }
}

/*************************************************************
Name:   Pa.java
Description:   Parse a WFF into a vector.
Data: January 2005
*************************************************************/

package MS;

import java.util.Vector;
import java.lang.Object.*;
import java.util.StringTokenizer;
import java.io.*;

public class Pa
{
   private String input="";
   private String temp="";
   private String b="";
   private String extracomma="";
   private char comma=',';
   String var1="";
   String var2="";
   Vector v = new Vector();
   Vector qv = new Vector();
   int count3=0, count2 =0, not = 0 ;


   public void print()
   {
        for( int i=0; i<v.size();i++){
          String s = (String) v.elementAt(i);
          System.out.println(s);
          }
    } //end print

  public Vector Pa( String s) throws Exception {
       this.input=s;
       int len = input.length();
       for( int i=0; i<len ; i++){
          if( input.charAt(i)==' ') b="";
          else if (input.charAt(i)=='(') {
            b=b + input.charAt(i) ;
             v.addElement(b);
             b="";
          }

          else if( input.charAt(i) ==')'){
             b=b+ input.charAt(i);
             System.out.println(b);
```

```
    v.addElement(b) ;
    b="";
}

else if ( input.charAt(i) == '\u00AC'){
    b=b+input.charAt(i) ;
    System.out.println(b);
    v.addElement(b) ;
    b="";
  } // not

else if (input.charAt(i) =='\u2227' || input.charAt(i)=='\u2228'
||input.charAt(i)=='\u2192'||input.charAt(i)=='\u2194'||
  input.charAt(i)=='\u2260'||input.charAt(i)=='=') {
     b=b+ input.charAt(i) ;
     v.addElement(b) ;
     b="";
 } //logical and||or||if then||if and only if|| not equal to|| '='

else if( input.charAt(i) =='\u2200' || input.charAt(i) =='\u2203' ||
         input.charAt(i)
      =='1'){
   //ForAll|| there exists || '1')
    temp += input.charAt(i) ;
    i=i+1;
    //getting the variables of the Quantifier.
    if( Character.isLowerCase(input.charAt(i))){
       temp += input.charAt(i) ;  i=i+1;count3=2;
       var1 += input.charAt(i);
       count3 = count3 - count3 ;
       if(input.charAt(i) == ' ') i=i;
       else i=i-1;

     } //end if lowercase
         else throw new Exception("2wrong");
         i=i+1;
         if(input.charAt(i) ==','){
              temp += input.charAt(i) ;
              i=i+1;
               if( Character.isLowerCase(input.charAt(i))){
                   temp += input.charAt(i) ;  i=i+1;count3=2;
                   var2 += input.charAt(i);
                   count3 = count3 - count3 ;
                   if(input.charAt(i) == ' ') i=i;
                   else i=i-1;
             } //end if lowercase
         else throw new Exception("2wrong");
         } // if ','
           else i=i-1;
           if(var1.equals(var2)) throw new Exception("wrong");
       } // end if loop Quantifier
 //getting the Predicate
 else if( Character.isUpperCase(input.charAt(i)){
       temp += input.charAt(i) ;  i=i+1;
      //If the Predicate has more than one Capital letter
       if (Character.isUpperCase(input.charAt(i) )) {
          temp += input.charAt(i) ; //++i ;
          i=i+1; count2= 0 ;
          while( Character.isUpperCase(input.charAt(i))) {
                if(count2<2) temp += input.charAt(i) ; //++i ;
                else throw new Exception("Wrong");
                count2=count2 +1; i=i+1 ;
              }
            count2=count2-count2;
            if (input.charAt(i)=='('){
               temp += input.charAt(i) ; //++i ;
               i=i+1;
```

```
                    //getting the variables
                    if ( Character.isLowerCase(input.charAt(i))){
                         temp+= input.charAt(i);
                         i=i+1 ;
                         if (input.charAt(i)==','){
                              if (input.charAt(i)==')'){
                                   temp += input.charAt(i) ; //++i ;
                                   }
                              else throw new Exception("3wrong");
                             }// Lower Case if
                        else throw new Exception("4wrong");
                    } //end if (
                    else throw new Exception("5wrong");
               } //end if uppercase

  else{ // if the Predicate has only one capital letter.
                  if (input.charAt(i)=='('){
                    temp += input.charAt(i) ; //++i ;
                    i=i+1;
                    //getting the variables
                    if( Character.isLowerCase(input.charAt(i))){
                          temp+= input.charAt(i);
                           i=i+1 ; //int count2 =0 ;
                           while(Character.isLowerCase(input.charAt(i))){
                               if(count2<2) temp += input.charAt(i) ; //++i ;
                               else throw new Exception("Wrong");
                               count2=count2+1; i=i+1 ;
                           }
                           count2=count2-count2 ;
                          if ((comma+"").equals(input.charAt(i)+"")) {
                                temp+= input.charAt(i);
                                i=i+1 ;
                                if( Character.isLowerCase(input.charAt(i))) {
                                     temp+= input.charAt(i);
                                     i=i+1 ;
                                     while( Character.isLowerCase(input.charAt(i))){
                                        if(count2<2){
                                        temp += input.charAt(i) ; //++i ;
                                        }
                                         else throw new Exception("Wrong");
                                         count2=count2+1; i=i+1 ;
                                         }
                                     count2=count2-count2 ;
                                    }// if lowercase
                                  }
                                  if (input.charAt(i)==')'){
                                    temp += input.charAt(i) ; //++i ;
                                  }
                                  else throw new Exception("55wrong");
                           }// Lower Case if
                          else throw new Exception("6wrong");
                    } //end if (
                    else throw new Exception("7wrong");
               }

          } // end else if uppercase

     if( temp != "") {
     temp="";

}// end for len loop

   if( not == 1) {
     v.addElement(")");
    }

   return v ;
```

```
   }//end Pa class

}// end Pa

*****************************************************************
Name:   BuildTree.java
Description:   Build a binary tree from a vector
Data: January 2006
*****************************************************************/


package MS;

import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class BuildTree {

     private String input="";
     Vector vec =new Vector();
     public BinaryTree BuildTree( Vector v  ) throws Exception {

                BinaryTree t = new BinaryTree();
                ChildNode root = new ChildNode();
                ChildNode temp = new ChildNode();
                ChildNode n = new ChildNode();
                ChildNode r = new ChildNode();
                ChildNode l = new ChildNode();
                ChildNode tempParent = new ChildNode();
                ChildNode tempRight = new ChildNode();
                t.setRoot(root);
                temp=root;
                Vector HoldQ = new Vector(); //Hold quantifiers
                Vector HoldP = new Vector(); //Hold predicates
                Vector HoldNotQ = new Vector(); //Hold quantifiers' negation sign
                Vector HoldNotP = new Vector(); //Hold predicate's negation sign
                Vector HoldNotC = new Vector(); // Hold Operator's negation sign
                String temp_op = "";
                String temp_predicate="";
                String temp_var1="";
                String temp_var2="";
                String temp_not="";
                String HoldPP="" ;
                String HoldVv="" ;
                String HoldDD="" ;
                String empty ="";
                String b= "";
                String a= "";
                boolean opnull =false;
                boolean commaLoop=false;
                int opO=0, neg = 0, LP = 0; x =0;
                int j=0,jj =0;y =0; ii =0 ; countQ =0;hq = 0;
                int notC =0; vector_used =0;newNode = 0;
                int QuantC=0;

                for(int i=0; i<v.size(); i=i+1)
                {
                    String s = (String) v.elementAt(i);
                    this.input = s ; x =0 ;
                    int len = s.length() ;

                    //if the character is a Quantifier
                    if(input.charAt(x)=='\u2200'||input.charAt(x)=='\u2203'
                    ||input.charAt(x)=='1')
                    //\u2200= 'For All' and \u2203= 'There Exist'
```

```
{   QuantC=QuantC+1;
    /*counting how many characters from vector 'v' are being
    used.*/
    vector_used=vector_used +1 ;
    countQ =countQ +1;
    n=temp;
    HoldQ.addElement(s);
    // add quantifier in temp vector
    if(input.charAt(x) =='\u2200')
        HoldQ.addElement('\u2200'+""); // add 'For All'
    if(input.charAt(x) == '\u2203')
        HoldQ.addElement('\u2203'+""); // add 'There Exist'
    if(input.charAt(x) == '1')
        HoldQ.addElement('1'+""); // add '1'
    x=x+1;
    //getting the varibales
    if( Character.isLowerCase(input.charAt(x)) ){
        t.pretrav();
        temp_var1 += input.charAt(x) ;  //x=x+1;
        int count3=2;
        HoldQ.addElement(temp_var1);
        temp_var1="";

        if(s.length()>3){
           x=x+1;
           if((',"+"").equals(input.charAt(x)+"")) {
              x=x+1;
                 if( Character.isLowerCase(input.charAt(x)) )
                 {
                     temp_var2 += input.charAt(x) ;  x=x+1;
                     HoldQ.addElement(temp_var2);
                     temp_var2="";
                 } //end if lowercase

            }//end comma
          } // s.lenght()
       else HoldQ.addElement("*");
       } //end if lowercase
       //setting the quatifier's negation sign
       if(neg==1) HoldNotQ.addElement("not");
       else  HoldNotQ.addElement("pos");
       neg = 0 ;
       x=0;
} // end if Quantifier
  // if  character is a negation sign, set neg.
else if( input.charAt(x)=='\u00AC')
{   vector_used=vector_used +1 ;
     if(LP==0) neg =1 ;
     else  neg = 1 ;
} //'\u00AC'=negation

else if( input.charAt(x)=='(')
{
      if(HoldQ.size()>0&&i>2){
         t.insertLeftGo();
       }
      if(HoldQ.size()> 0) {
      while(HoldQ.size()>0){
          // if there is Quantifier stored in HoldQ
          t.inData((String)HoldQ.elementAt(0));
          t.inQuantifier((String)HoldQ.elementAt(1));
          t.inVariable1((String)HoldQ.elementAt(2));
          t.inVariable2((String)HoldQ.elementAt(3));
          // Set Quantifier to appropriate negation sign.
           if(HoldNotQ.size() >0){
             if((String)HoldNotQ.elementAt(0) == "not")
              {t.setNot();
```

```
            }
            HoldNotQ.removeElementAt(0);
          }// empty HoldQ
          HoldQ.removeElementAt(3);
          HoldQ.removeElementAt(2);
          HoldQ.removeElementAt(1);
          HoldQ.removeElementAt(0);
          t.insertRightGo();
        }
      }
      else {
        t.insertLeftGo();
      }// HoldQ.size < 1
      if(QuantC==1)
        QuantC=QuantC+1;
        vector_used=vector_used +1 ;
      //setting the negation sign
      if(neg == 1){
       notC=notC +1 ;
       HoldNotC.addElement("not") ;
      }
      else {
        HoldNotC.addElement("pos") ;
        neg = 0;
      }
}
// if Character is a predicate
else if( Character.isUpperCase(input.charAt(x))){
  vector_used=vector_used +1 ;
  temp_predicate += input.charAt(x);
  x=x+1;
  if (Character.isUpperCase(input.charAt(x))){
        x=x+1 ;
        while( Character.isUpperCase(input.charAt(x) )) {
          temp_var1 += input.charAt(x) ;
          i=i+1;
         }//getting the predicate
    } //if

   if (input.charAt(x) == '(') {
       x=x+1 ;
       //getting the variables
       if ( Character.isLowerCase(input.charAt(x))) {
           temp_var1 += input.charAt(x) ;
           x=x+1 ;
           if ((',+"").equals(input.charAt(x)+"")) {
                 x=x+1 ;
                 if( Character.isLowerCase(input.charAt(x))) {
                 temp_var2 += input.charAt(x);
                 x=x+1 ;
                 } //if lower case
              } //if comma

            else  temp_var2="*";
           } //if
         }//if Lowercase
 // Putting the predicate in a vector called HoldP
  HoldP.addElement(s);
  HoldP.addElement(temp_predicate) ;
  HoldP.addElement(temp_var1);
  HoldP.addElement(temp_var2);
  /*putting the corresponding negation sign in the HoldNotP
   vector.*/
    if(neg == 0)
      HoldNotP.addElement("pos");
    else
      HoldNotP.addElement("not");
```

```
    neg=0;
  temp_predicate ="";
  temp_var1="";
  temp_var2="";
//If we are reaching the end of vector 'v'
if((vector_used+3)>v.size()) {
  // if HoldP in not zero fill the ChildNode.
   if(HoldP.size() > 1 ) {
      if( HoldQ.size() > 0) {
         while(HoldQ.size() > 0) {
            t.inData((String)HoldQ.elementAt(0)); S
            t.inQuantifier((String)HoldQ.elementAt(1));
            t.inVariable1((String)HoldQ.elementAt(2));
            t.inVariable2((String)HoldQ.elementAt(3));
            if(HoldNotQ.size()>0) {
                 if((String)HoldNotQ.elementAt(0)=="not")
                    { t.setNot() ;
                      HoldNotQ.removeElementAt(0);
                    }
             }
             HoldQ.removeElementAt(HoldQ.size()-3);
             HoldQ.removeElementAt(HoldQ.size()-2);
             HoldQ.removeElementAt(HoldQ.size()-1);

             if(HoldQ.size()<2) HoldQ.removeElementAt(0);
             else HoldQ.removeElementAt(HoldQ.size());
             t.insertLeftGo();
             }
       }
          t.inData((String)HoldP.elementAt(0));
          t.inPredicate((String)HoldP.elementAt(1));
          t.inVariable1((String)HoldP.elementAt(2));
          t.inVariable2((String)HoldP.elementAt(3));
             if(HoldNotP.size()>0) {
                 if((String)HoldNotP.elementAt(0)=="not")
                     t.setNot() ;
                  HoldNotP.removeElementAt(0);
              }
          // empty HoldP
          HoldP.removeElementAt(3);
          HoldP.removeElementAt(2);
          HoldP.removeElementAt(1);
          HoldP.removeElementAt(0);
          t.pretrav();
          //t.getParent() ;
  }
 }
}// end else if for Uppercase
// if the character is a operator
else if(input.charAt(x)=='\u2227'||input.charAt(x)=='\u2228'||
        input.charAt(x)=='\u2192'||input.charAt(x)=='\u2194'||
        input.charAt(x)=='\u2260'||input.charAt(x)=='=') {
        /*Operators('and'||'or'||'if then'||
          'if and only if'||'not equal'|| '=')*/
         opO =opO+1;
         if(t.getData() == null) {
         vector_used=vector_used +1 ;
         newNode = newNode - newNode ;
         t.insertLeftGo();
         int pl = 0;
         while((LP-1) > pl) {
            pl=pl+1;
         }
        //insert Predicate from HoldP into the ChildNode
        if(HoldP.size() > 1){
            while( HoldQ.size() != 0){
              t.inData((String)HoldQ.elementAt(0));
```

```
                t.inQuantifier((String)HoldQ.elementAt(1));
                t.inVariable1((String)HoldQ.elementAt(2));
                t.inVariable2((String)HoldQ.elementAt(3));
                if(HoldNotQ.size()>0) {
                   if((String)HoldNotQ.elementAt(0)=="not")
                      t.setNot() ;
                     HoldNotQ.removeElementAt(0);
                }
                HoldQ.removeElementAt(3);
                HoldQ.removeElementAt(2);
                HoldQ.removeElementAt(1);
                HoldQ.removeElementAt(0);
                t.insertLeftGo();
                hq=hq+1 ;
               }
     t.inData((String)HoldP.elementAt(0));
     t.inPredicate((String)HoldP.elementAt(1));
     t.inVariable1((String)HoldP.elementAt(2));
     t.inVariable2((String)HoldP.elementAt(3));
     if(HoldNotP.size() > 0){
         if((String)HoldNotP.elementAt(0) == "not"){
           t.setNot();
           HoldNotP.removeElementAt(0);
         }
         else HoldNotP.removeElementAt(0);
       }
    // empty out HoldP
     HoldP.removeElementAt(3);
     HoldP.removeElementAt(2);
     HoldP.removeElementAt(1);
     HoldP.removeElementAt(0);

     if(t.getData() != null){
       while(t.getData() != null){
             t.getParent();
       }
     }
     LP= LP-LP;
     t.inData(input.charAt(x)+"");
     if(input.charAt(x) == '\u2227')//'and'
        t.inOperator("\u2227");
     else if (input.charAt(x) == '\u2228')//'or'
        t.inOperator("\u2228");
     else if (input.charAt(x) == '\u2192')//'if then'
        t.inOperator("\u2192");
     else if (input.charAt(x) == '\u2260')//'not equal'
        t.inOperator("\u2260");
     else if (input.charAt(x) == '=')
        t.inOperator("=");
     else t.inOperator("\u2194");//'if and only if'
     // setting the negation sign for the Operators
     if(HoldNotC.size() >0 ){
           if((String)HoldNotC.elementAt(0)=="not"){
            t.setNot() ;
            HoldNotC.removeElementAt(0);
           }
           else HoldNotC.removeElementAt(0);
        }
         t.insertRightGo();
}//end if no HoldP >0
else {
   if(t.getRoot() != t.getCurrent()){
      while(t.getData() != null)
            t.getParent();
    }
    if(t.getRoot() == t.getCurrent()){
       t.insertParentGo();
```

```
                t.inData(input.charAt(x)+"");
                if(input.charAt(x) == '\u2227')//'and'
                     t.inOperator("\u2227");
                else if (input.charAt(x) == '\u2228')//'or'
                     t.inOperator("\u2228");
                else if (input.charAt(x) == '\u2192')//'if then'
                     t.inOperator("\u2192");
                else if (input.charAt(x) == '\u2260')//'not equal'
                     t.inOperator("\u2260");
                else if (input.charAt(x) == '=')
                     t.inOperator("=");
                else t.inOperator("\u2194");//'if and only if'

              if(HoldNotC.size() >0 ){
                 if((String)HoldNotC.elementAt(0)=="not"){
                    t.setNot() ;
                    HoldNotC.removeElementAt(0);
                   }
                   else
                     HoldNotC.removeElementAt(0);
                 }
            }
          }
         } // get left is null
         else {
               opnull =false ;
               while( opnull == false) {
                   if(t.getCurrent() == t.getRoot()) {
                         if(t.getData() == null) opnull=true;
                         else {
                         t.insertParentGo();
                         t.setRoot(t.getCurrent());
                         opnull =true;
                         }
                        }
                   if( t.getData() == null) opnull =true;
                   else t.getParent();
               }
               if(input.charAt(x) == '\u2227'){//'and'
                  t.inOperator("\u2227");
                  t.inData("\u2227");
               }
               else if (input.charAt(x) == '\u2228'){ //'or'
                  t.inOperator("\u2228");
                  t.inData("\u2228");
               }
               else if (input.charAt(x) == '\u2192'){ //'if then'
                  t.inOperator("\u2192");
                  t.inData("\u2192");
               }
               else if (input.charAt(x) == '\u2260'){ //'not equal'
                  t.inOperator("\u2260");
                  t.inData("\u2260");
               }
               else if (input.charAt(x) == '=') {
                  t.inOperator("=");
                  t.inData("=");
               }
               else {
                  t.inOperator("\u2194"); //'if and only if'
                  t.inData("\u2194");
               }
               if(HoldNotC.size() >0 ) {
                  if((String)HoldNotC.elementAt(0)=="not"){
                     t.setNot() ;
                     HoldNotC.removeElementAt(0);
                    }
```

```
                    else HoldNotC.removeElementAt(0);
                  }
              }
            } // end operator loop

            else if( input.charAt(x)==')')
            { // if there are predicate left
              if(HoldP.size() > 1 ) {
                //if there are Quantifiers left
                if( HoldQ.size() > 0){

                   while(HoldQ.size() > 0) {
                      t.inData((String)HoldQ.elementAt(0));
                      t.inQuantifier((String)HoldQ.elementAt(1));
                      t.inVariable1((String)HoldQ.elementAt(2));
                      t.inVariable2((String)HoldQ.elementAt(3));
                         if(HoldNotQ.size()>0) {
                             if((String)HoldNotQ.elementAt(0)=="not") {
                                     t.setNot() ;
                                 HoldNotQ.removeElementAt(0);
                               }
                           }
                      HoldQ.removeElementAt(0);
                      HoldQ.removeElementAt(0);
                      HoldQ.removeElementAt(0);
                      if(HoldQ.size()<1) HoldQ.removeElementAt(0);
                      else if(HoldQ.size()<6) HoldQ.removeElementAt(0);
                      else HoldQ.removeElementAt(0);S
                      t.insertLeftGo();
                        }
                    }
                      t.inData((String)HoldP.elementAt(0));
                      t.inPredicate((String)HoldP.elementAt(1));
                      t.inVariable1((String)HoldP.elementAt(2));
                      t.inVariable2((String)HoldP.elementAt(3));
                         if(HoldNotP.size()>0){
                             if((String)HoldNotP.elementAt(0)=="not") [
                                t.setNot() ;
                                }
                              HoldNotP.removeElementAt(0);
                             }
                      HoldP.removeElementAt(3);
                      HoldP.removeElementAt(2);
                      HoldP.removeElementAt(1);
                      HoldP.removeElementAt(0);
                      t.pretrav();
                      t.getParent() ;
              }
              else {
                   if(t.getData() != null){
                      commaLoop=true;
                      while(commaLoop==true){
                            if((t.getRoot()!= t.getCurrent())&&(t.getData()
                                  != null)){
                               commaLoop=true;
                               t.getParent();
                            }
                            else commaLoop=false;
                      }
                  }
              }
            }// end if ")"

          }//end for v.size()
              //if there are Predicates left
               if(HoldP.size() > 1){
                          while( HoldQ.size() != 0){
```

```
                                    t.inData((String)HoldQ.elementAt(0));
                                    t.inQuantifier((String)HoldQ.elementAt(1));
                                    t.inVariable1((String)HoldQ.elementAt(2));
                                    t.inVariable2((String)HoldQ.elementAt(3));
                                    if(HoldNotQ.size()>0){
                                       if((String)HoldNotQ.elementAt(0)=="not")
                                           t.setNot() ;
                                       HoldNotQ.removeElementAt(0);
                                       }
                                    HoldQ.removeElementAt(3);
                                    HoldQ.removeElementAt(2);
                                    HoldQ.removeElementAt(1);
                                    HoldQ.removeElementAt(0);
                                    t.insertLeftGo();
                                    hq=hq+1 ;
                                   }
                                t.inData((String)HoldP.elementAt(0));
                                t.inPredicate((String)HoldP.elementAt(1));
                                t.inVariable1((String)HoldP.elementAt(2));
                                t.inVariable2((String)HoldP.elementAt(3));
                                if(HoldNotP.size() > 0){
                                  if((String)HoldNotP.elementAt(0) == "not"){
                                    t.setNot();
                                    HoldNotP.removeElementAt(0);
                                  }
                                  else HoldNotP.removeElementAt(0);
                                }
                               HoldP.removeElementAt(3);
                               HoldP.removeElementAt(2);
                               HoldP.removeElementAt(1);
                               HoldP.removeElementAt(0); t.pretrav();
                              t.pretrav();
                            }
            t.setCurrent(t.getRoot());
            return t ;
     }// end public void BuildTree

}// end BuildTree class

*************************************************************
Name:   CheckParse.java
Description: Translate the input string. In particular translate
the unicode to english to make sure the user inputed the correct
String.
Data: January 2006
*************************************************************/
package MS;

import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class CheckParse{

   private String input="";
   private String temp="";
   private String b="";
   private String extracomma="";
   Vector v = new Vector();

    public CheckParse(){}

    public String CheckParse( String s) throws Exception {
     this.input=s;
     int len = input.length();
```

```
    for( int i=0; i<len ; i++) {
        if(input.charAt(i)=='\u22A2') {
             temp=temp+" implies " ;
             b="";
         }
         else if ( input.charAt(i) == '\u00AC')
            {b=b+input.charAt(i) ;
             System.out.println(b);
             temp = temp+" Not ";
             b="";
            } // not
         else if (input.charAt(i) =='\u2227')
            {b=b+ input.charAt(i) ;
             temp=temp +" And ";
             b="";
             } //logical and
         else if (input.charAt(i)=='\u2228')
            {b=b+ input.charAt(i) ;
             temp=temp+" Or " ;
             b="";
             } //logical or
         else if (input.charAt(i)=='\u2192')
            {b=b+ input.charAt(i) ;
             temp=temp+" If then " ;
             b="";
             } //logical If Then
         else if (input.charAt(i)=='\u2194')
            {b=b+ input.charAt(i) ;
             temp=temp+" If and only if " ;
             b="";
             } //logical If and only if
         else if (input.charAt(i)=='\u2260')
            {b=b+ input.charAt(i) ;
             temp=temp+" Not equal " ;
             b="";
             } //not equal
         else if(input.charAt(i) =='\u2200')
             {
             b=b+ input.charAt(i) ;
             temp=temp+" For All " ;
             b="";
             } //logical For All
         else if(input.charAt(i) =='\u2203')
            {b=b+ input.charAt(i) ;
             temp=temp+" There Exists ";
             b="";
             } //logical There Exists
          else if(input.charAt(i) != ' ')
           { b=b+ input.charAt(i) ;
             temp=temp+ b ;
             b="";
           }
       }// for loop
        return temp ;
    }// CheckParse

}// Class CheckParse

/*************************************************************
Name:   CopyTree.java
Description:   Based on ParseTree.java. This is to copy a Binary Tree
Data: January 2006
*************************************************************/

package MS;
```

```
import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class CopyTree{

  public BinaryTree CopyTree( BinaryTree d,BinaryTree c) {

      ChildNode start = new ChildNode() ;
      ChildNode startc = new ChildNode() ;
      ChildNode opRoot = new ChildNode();
      String pt = "" ;
      String vari ="";
      String quant = "";
      String predi = "";
      String oper = "";
      String pred="";
      String rootS="";
      boolean go = true;
      boolean crossroot=false;//did the program cross the root.
      boolean goingright=false;//did the program go to the right side of the tree.
      boolean opRootCheck=true;/* to help identify the operator at the top of the parse
      tree.*/
      boolean loop=true;
      int goingrightcount=0; //how many times did the program go right.
      int goingrighthelp=0;
      int goingup = 0; //helps identify if the program is going up the parse tree.
      int startOP =0; /*identifies if the first item the program identifies is an
      operator.*/
      int QuantP = 0; //counts how many quantifiers so far.
      int count = 0;
      int count1OP = 0; //counts and keeps track of the operators.
      int count1Q = 0; /*identifies if the first item the program identifies is a
      quantifier.*/
      int count2Q = 0;//identifies how many quantifiers in while loop.
      int i = 0 ,int jj =0;
      int jjhelp=0;
      int stopLoop=15; // to help prevent endless loops.

         startc=c.getCurrent();
         start = d.getCurrent();
         rootS = d.getCurrent().getData();

         if(d.getQuantifier() != null) {
         if(d.getNot() == true)c.setNot() ;
         if(d.getLnot() == true)c.setLNot();
         if(d.getRnot() == true)c.setRNot();
         if(d.getCenter() == true)c.setCenter();
         c.inData(d.getData());
         c.inQuantifier(d.getQuantifier());
         c.inVariable1(d.getVariable1());
         if(d.getVariable2() != null)
         c.inVariable2(d.getVariable2());
         c.inLeftData(d.getLeftData());
         c.inRightData(d.getRightData());

          if(d.getCurrent().getRight() != null){
              d.getRight();
              c.insertRightGo();
            }
           else {
              d.getLeft();
              c.insertLeftGo();
          }
           while(d.getQuantifier() != null){
            if(d.getNot() == true) c.setNot() ;
```

```
        if(d.getLnot() == true) c.setLNot();
        if(d.getRnot() == true) c.setRNot();
        if(d.getCenter() == true) c.setCenter();
        c.inData(d.getData());
        c.inQuantifier(d.getQuantifier());
        c.inVariable1(d.getVariable1());
        if(d.getVariable2() != null)
        c.inVariable2(d.getVariable2());
        c.inLeftData(d.getLeftData());
        c.inRightData(d.getRightData());
        count1Q = count1Q + 1;
        d.getRight();
        c.insertRightGo();
     }
     QuantP += QuantP;
  }
  else if (d.getOperator() != null) {
          startOP =1;
          if(d.getNot() == true) c.setNot() ;
          if(d.getLnot() == true) c.setLNot();
          if(d.getRnot() == true) c.setRNot();
          if(d.getCenter() == true) c.setCenter();
          c.inData(d.getData());
          c.inOperator(d.getOperator());
          c.inLeftData(d.getLeftData());
          c.inRightData(d.getRightData());
          opRootCheck=false;
          opRoot=d.getCurrent();
          d.getLeft();
          c.insertLeftGo();
          count1OP = count1OP +1 ;
  }

  else if (d.getPredicate() != null) {
         if(d.getNot() == true) c.setNot() ;
         if(d.getLnot() == true) c.setLNot();
         if(d.getRnot() == true) c.setRNot();
         if(d.getCenter() == true) c.setCenter();
         c.inData(d.getData());
         c.inPredicate(d.getPredicate());
         c.inVariable1(d.getVariable1());
         if(d.getVariable2() != null) c.inVariable2(d.getVariable2());
         c.inLeftData(d.getLeftData());
         c.inRightData(d.getRightData());
         i=i+14;
         d.setCurrent(start);
         return c;
}
else {}//do nothing

  while(i < stopLoop) {
    i = i +1 ;
   //copy the quantifier information
   if(d.getCurrent().getQuantifier() != null){
          if(d.getNot() == true) c.setNot() ;
          if(d.getLnot() == true) c.setLNot();
          if(d.getRnot() == true) c.setRNot();
          if(d.getCenter() == true) c.setCenter();
          c.inData(d.getData());
          c.inQuantifier(d.getQuantifier());
          c.inVariable1(d.getVariable1());
          if(d.getVariable2() != null) c.inVariable2(d.getVariable2());
          c.inLeftData(d.getLeftData());
          c.inRightData(d.getRightData());
          if(d.getQuantifier() =="0"){
              c.inData(d.getVariable1());c.pretrav();
              d.setCurrent(start);
```

```
            return c;
          }
        QuantP += QuantP;
        count2Q = count2Q +1;
        if(d.getCurrent().getRight() == null) {

           d.getLeft();
           if(c.getLeft() == null){
               c.insertLeftGo();
           }
            else {}
            c.getLeft();
    }
    else {
         d.getRight() ;
         c.insertRightGo();
    }
   }
// copy the operator information
 else if(d.getOperator() != null) {
        if(d.getNot() == true) c.setNot() ;
        if(d.getLnot() == true) c.setLNot();
        if(d.getRnot() == true) c.setRNot();
        if(d.getCenter() == true) c.setCenter();
        c.inData(d.getData());
        c.inOperator(d.getOperator());
        c.inVariable1(d.getVariable1());
        if(d.getVariable2() != null) c.inVariable2(d.getVariable2());
        c.inLeftData(d.getLeftData());
        c.inRightData(d.getRightData());
  if(opRootCheck == true) {
      opRoot=d.getCurrent();
      opRootCheck = false;
    }
 if(goingup == 1) {
     if( count1OP > 0) {
         while( count1OP != 0 ){
            d.getParent() ;
            c.getParent();
            count1OP = count1OP - 1 ;
         }// end while
         d.setCurrent(start);
         return c;
     }
   }
 else{
 count1OP = count1OP + 1 ;
 d.getLeft(); goingright=false;
 if(c.getCurrent().getLeft() == null)
        c.insertLeftGo();
     else
        c.getLeft();
   }
  } // end else goingup
 // Copy Predicate information
 else if(d.getPredicate() != null) {
    if(d.getNot() == true) c.setNot() ;
    if(d.getLnot() == true) c.setLNot();
    if(d.getRnot() == true)c.setRNot();
    if(d.getCenter() == true) c.setCenter();
    c.inData(d.getData());
    c.inPredicate(d.getPredicate());
    c.inVariable1(d.getVariable1());
    if(d.getVariable2() != null) c.inVariable2(d.getVariable2());
    c.inLeftData(d.getLeftData());
    c.inRightData(d.getRightData());
```

```
if(goingright==true){
   goingright = false;
   d.getParent();
   c.getParent();
/* After getting the number of times the program went right, adjust how may
times the program should go up the parse tree.*/
if(goingrighthelp == 4) jjhelp=2;
else jjhelp=1;
while( loop==true){
          if(d.getCurrent() == d.getRoot()) loop = false;
          //If it is a quantifier go up.
          if(d.getQuantifier() != null){
              d.getParent();
              c.getParent();
              if(d.getCurrent() == d.getRoot()) loop = false;
            }
          //If it is a operator test to see if you should stop the loop.
          else if(d.getOperator() != null){
                  if(jj==jjhelp){
                    loop=false;
                   }
                   else {
                     d.getParent();
                     c.getParent();
                      if(d.getCurrent() == d.getRoot()) loop =false;
                      jj=jj+1;
                      if(d.getData()==null)
                         loop=false;
                   }
          }
          else if(d.getData() == null){
            d.getParent();
            c.getParent();
            if(d.getCurrent() == d.getRoot()) loop =false;
          }
          else{ loop = false;}
         }
jj=jj-jj;
loop=true;
if(goingrighthelp == 4) goingrighthelp=0;
go=true;

while( go ==true){
   if( d.getOperator() != null){
       /*if the operator is the operator at the top of the parse tree, end
       program. else go right.*/
       if(d.getCurrent() == d.getRoot() || d.getCurrent()==opRoot){
           if(crossroot == true){
               d.setCurrent(start);
               return c;
             } // if crossroot == true
           else {
              go=false;
              crossroot=true;
              d.getRight();
              c.insertRightGo();
              goingright=false;

           }// crossroot==false
       }// d.getRoot != null
    else {
        go=false;
        goingright=true;
        goingrightcount=goingrightcount + 1;
        goingrighthelp=goingrighthelp +1;
        d.getRight();
        c.insertRightGo();
```

```
      } // d.getRoot == null
    }// if "d.getOperator()
    else if(d.getQuantifier() != null){
        /* if it is the root, end program. If it is not the root go to the
        parent node.*/
        if(d.getRoot() == d.getCurrent()){
            d.setCurrent(start);
            return c;
        }
        else{
            d.getParent();
            c.getParent();
        }//not root
    }//d.getQuantifier
    else {
       d.setCurrent(start);
       return c;
    }
   } // while "go" loop
}//end goingright
else {
    d.getParent();
    c.getParent();
    go=true;
    while(go ==true){
      if(d.getOperator() != null){
          go=true;
          while( go == true){
              if(d.getOperator() != null){
                 /* if the program has gone right at least once, subtract
                 goingrightcount.*/
                  if(goingrightcount >0){
                      goingrightcount = goingrightcount - 1;
                   }
                  if(goingrightcount == 0){
                      if(d.getCurrent() == d.getRoot()||
                         d.getCurrent()==opRoot){
                          if(crossroot == false){
                          //going to the right of the root.
                            crossroot = true;
                            go = false;
                            d.getRight();
                            c.insertRightGo();
                          }
                   else {
                       d.setCurrent(start);
                       c.setCurrent(startc);
                       return c;
                    }

              }// d.getRoot() != null
             else{
                 goingrighthelp=goingrighthelp+1;
                 goingrightcount = goingrightcount + 1;
                 goingright =true;
                 d.getRight();
                 c.insertRightGo();
                 go=false;

              } //d.getRoot() == null
           }// goingrightcount == 0
          else{
            go=true;
            d.getParent();
            c.getParent();
          }
```

```
                    }
                  }// go ==true
                }//d.getOperator !=null
                else if(d.getQuantifier() != null){
                     /* If it is the root end program else go to the parent node.*/
                          if(d.getRoot() == d.getCurrent() || d.getCurrent() == opRoot){
                              d.setCurrent(start);
                              return c;
                           }//d.getRoot() != null
                           else {
                              d.getParent();
                              c.getParent();
                              go=true;
                           } // d.getRoot() == null
                }//d.getQuantifier != null
                else{
                    d.setCurrent(start);
                    return c;
                }
              }//go =true
            }//end not goingright

            }//else if Predicate

            else{
                d.getLeft() ;
                goingright=false;
            }
          } // end while loop

    d.setCurrent(start);
    return c ;

   } //

}// class CopyTree

/*************************************************************
Name: CaptureTree.java
Description: Finding a suitable variable in a capture situation
Data: January 2006
*************************************************************/

package MS;


import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class CaptureTree{


  public String CaptureTree( BinaryTree d) {

      ChildNode start = new ChildNode() ;
      ChildNode opRoot = new ChildNode();
      BinaryTree t = new BinaryTree() ;
      Vector newT = new Vector() ;
      Vector Qvar = new Vector();
      String pt = "" ;
      String vari ="";
      String quant = "";
      String predi = "";
      String oper = "";
      String pred="";
```

```
String rootS="";
String var1="";
String var2="";
String Svar1="";
String Svar2="";
String finish="";
boolean go = true;
boolean crossroot=false;//did the program cross the root
boolean goingright=false;//did the program go right
boolean opRootCheck=true;//did the program identify the operator at the top of the
boolean loop=true;
int goingrightcount=0;// how many times did the program go right.
int goingrighthelp=0;
int goingup = 0 ;//helps identify if the program is going up the parse tree.
int startOP =0;/* identifies if the first item the program identifies is an
operator.*/
int QuantP = 0; // counts how many quantifiers so far.
int count = 0 ;
int count1OP = 0;//counts and keeps track of the operators.
int count1Q = 0;/*identifies if the first item the program identifies is a
quantifier.*/
int count2Q = 0;//identifies how many quantifiers in while loop.
int i = 0, ii=0,j = 0, jj =0, jjhelp=0,
int stoploop=15; //help prevent endless loops.

//Fill 'newT' with all the alphabets.
int stopLoop=15; //prevent endless loops
newT.addElement("a"); newT.addElement("b");
newT.addElement("c"); newT.addElement("d");
newT.addElement("e"); newT.addElement("f");
newT.addElement("g");newT.addElement("h");
newT.addElement("i"); newT.addElement("j");
newT.addElement("k");newT.addElement("l");
newT.addElement("m");newT.addElement("n");
newT.addElement("o"); newT.addElement("p");
newT.addElement("q");newT.addElement("r");
newT.addElement("s");newT.addElement("t");
newT.addElement("u"); newT.addElement("v");
newT.addElement("w");newT.addElement("x");
newT.addElement("y");newT.addElement("z");
start = d.getCurrent();
rootS = d.getCurrent().getData();

System.out.println("**********CaptureTree^^^^^^^^^^");

d.setCurrent(start);

  if(d.getQuantifier() != null) {
    if( d.getNot() == true)  pt = pt + "\u00AC" +d.getData();
    else  pt = pt +d.getData();
    count1Q = count1Q + 1;
    //identify Svar1 and Svar2 as the capture variable.
    Svar1=d.getVariable1();
    Svar2=d.getVariable2();

    if(d.getCurrent().getRight() != null){
        System.out.println(d.getData()+"quantstart2");
        d.getRight();
      }
     else d.getLeft();

     while(d.getQuantifier() != null){
        if( d.getNot() == true)  pt = pt + "\u00AC" +d.getData();
        else  pt = pt + d.getData();
        count1Q = count1Q + 1;
        //identify Svar1 and Svar2 as the capture variable.
        Svar1=d.getVariable1();
```

```
        Svar2=d.getVariable2();
        var1=d.getVariable1();
        var2=d.getVariable2();
        Qvar.addElement(var1);
        Qvar.addElement(var2);
        d.getRight();
     }
     QuantP += QuantP;
  }
  else if (d.getOperator() != null) {
     startOP =1;
     opRootCheck=false;
     opRoot=d.getCurrent();
     d.getLeft();
     count1OP = count1OP +1 ;
  }

  else if (d.getPredicate() != null) {
     i=i+14;
     d.setCurrent(start);
     while( ii < newT.size()){
         while(j < Qvar.size()){
            if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
               ii=ii+1;
            else{
               if(j == Qvar.size() -1){
                  finish = (String)newT.elementAt(ii);
                  ii=ii+100;
                  j=j+100;
                }
               else j=j+1;
            }
         }
     }
     return finish;
}
else {}

  while(i < stopLoop) {
   i = i +1;

   if(d.getCurrent().getQuantifier() != null){
     QuantP += QuantP;
     count2Q = count2Q +1 ;
     /* "0" quantifier is only a place holder for "=" operator.*/
     if(d.getQuantifier() == "0"){
        while( ii < newT.size()){
          while(j < Qvar.size()){
            if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
                ii=ii+1;
            else{
               if(j == Qvar.size() -1){
                  finish = (String)newT.elementAt(ii);
                  ii=ii+100;
                  j=j+100;
                }
               else j=j+1;
            }
         }
     }
        return finish;
     }
     if(d.getCurrent().getRight() == null) {
          var1=d.getVariable1();
          var2=d.getVariable2();
          if(Svar1.equals(var1)|| Svar2.equals(var1)){
            //do nothing
```

```
        }
        else
          Qvar.addElement(var1);
        if(Svar1.equals(var2)|| Svar2.equals(var2)){
          //do nothing
        }
        else
          Qvar.addElement(var2);
        d.getLeft();
    }
    else {
      var1=d.getVariable1();
      var2=d.getVariable2();
        if(Svar1.equals(var1)|| Svar2.equals(var1)){
          //do nothing
        }
        else
          Qvar.addElement(var1);
          if(Svar1.equals(var2)|| Svar2.equals(var2)){
          //do nothing
        }
        else
          Qvar.addElement(var2);
          d.getRight();
    }

   }

  else if(d.getOperator() != null) {
   if(opRootCheck == true) {
      opRoot=d.getCurrent();
      opRootCheck = false;
    }

 if(goingup == 1) {
    pt=pt+'(';
    if( count1OP > 0) {
      while( count1OP != 0 ){
         d.getParent() ;
         count1OP = count1OP - 1 ;
        }// end while
       d.setCurrent(start);

       while( ii < newT.size()){
       while(j < Qvar.size()){
          if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
             ii=ii+1;
          else{
             if(j == Qvar.size() -1){
                finish = (String)newT.elementAt(ii);
                ii=ii+100;
                j=j+100;
              }
             else j=j+1;
          }
       }
   }
       return finish ;
      }

    }
  else {
    count1OP = count1OP + 1 ;
    d.getLeft(); goingright=false;
    }
   } // end else goingup
```

```
else if(d.getPredicate() != null) {
   pred=d.getData();
   if(goingright==true){
     goingright = false;
     d.getParent();
   if(goingrighthelp == 4)
     jjhelp=2;
   else
     jjhelp=1;

   while( loop==true){
         if(d.getCurrent() == d.getRoot()) loop = false;
         if(d.getQuantifier() != null){
                 d.getParent();
                 if(d.getCurrent() == d.getRoot()) loop = false;
               }
             else if(d.getOperator() != null){
                     if(jj==jjhelp){
                       loop=false;
                      }
                      else {
                        d.getParent();
                         if(d.getCurrent() == d.getRoot()) loop =false;
                         jj=jj+1;
                      }
             }
             else{
               d.getParent();
               if(d.getCurrent() == d.getRoot()) loop =false;
             }
            }
      jj=jj-jj;
      loop=true;

      if(goingrighthelp == 4)
        goingrighthelp=0;

   go=true;
   while( go ==true){
    if( d.getOperator() != null){
       if(d.getCurrent() == d.getRoot() || d.getCurrent()==opRoot){
          if(crossroot == true){
             d.setCurrent(start);
             while( ii < newT.size()){
     while(j < Qvar.size()){
        if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
           ii=ii+1;
        else{
           if(j == Qvar.size() -1){
              finish = (String)newT.elementAt(ii);
              ii=ii+100;
              j=j+100;
            }
           else j=j+1;
        }
     }
}
             return finish;
           } // if crossroot == true
          else {
             go=false;
             crossroot=true;

             d.getRight();
             goingright=false;

           }// crossroot==false
```

```
        }// d.getRoot != null
       else {
          go=false;
          goingright=true;
          goingrighthelp=goingrighthelp+1;
          goingrightcount=goingrightcount + 1;
          pt=pt+d.getData();
          if(d.getCurrent().getRight() != null)
             d.getRight();
          else {
             return finish;
          }
       } // d.getRoot == null
    }// if "d.getOperator()
   else if(d.getQuantifier() != null){
       if(d.getRoot() == d.getCurrent()){
          d.setCurrent(start);
          while( ii < newT.size()){
    while(j < Qvar.size()){
       if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
          ii=ii+1;
       else{
          if(j == Qvar.size() -1){
             finish = (String)newT.elementAt(ii);
             ii=ii+100;
             j=j+100;
           }
          else j=j+1;
       }
    }
}
          return finish;
       }
       else{
          d.getParent();
       }//not root
   }//d.getQuantifier
   else {
      d.setCurrent(start);
      while( ii < newT.size()){
    while(j < Qvar.size()){
       if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
          ii=ii+1;
       else{
          if(j == Qvar.size() -1){
             finish = (String)newT.elementAt(ii);
             ii=ii+100;
             j=j+100;
           }
          else j=j+1;
       }
    }
}
      return finish;
   }
  } // while "go" loop
}//end goingright
else {
  d.getParent();
  go=true;
  while(go ==true){
    if(d.getOperator() != null){

      go=true;
      while( go == true){
        if(d.getOperator() != null){
          if(goingrightcount >0){
```

```
            goingrightcount = goingrightcount - 1;
            }
          if(goingrightcount == 0){

              if(d.getCurrent() == d.getRoot()|| d.getCurrent()==opRoot){
               if(crossroot == false){
                 crossroot = true;
                 go = false;
                 d.getRight();
                }
               else {
                 d.setCurrent(start);
                 while( ii < newT.size()){
    while(j < Qvar.size()){
       if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
          ii=ii+1;
       else{
          if(j == Qvar.size() -1){
             finish = (String)newT.elementAt(ii);
             ii=ii+100;
             j=j+100;
           }
          else j=j+1;
       }
    }
}
                 return finish;
               }

              }// d.getRoot() != null
             else{
               goingrighthelp=goingrighthelp+1;
               goingrightcount = goingrightcount + 1;
               goingright =true;
               pt=pt+d.getData();
               d.getRight();
               go=false;
              } //d.getRoot() == null
           }// goingrightcount == 0
          else{
            go=true;
            d.getParent();
           }
          }
        }// go ==true
    }//d.getOperator !=null
    else if(d.getQuantifier() != null){
            if(d.getRoot() == d.getCurrent() || d.getCurrent() == opRoot){
               d.setCurrent(start);
                 while( ii < newT.size()){
    while(j < Qvar.size()){
       if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
          ii=ii+1;
       else{
          if(j == Qvar.size() -1){
             finish = (String)newT.elementAt(ii);
             ii=ii+100;
             j=j+100;
           }
          else j=j+1;
       }
    }
}
            return finish ;
            }//d.getRoot() != null
            else {
               d.getParent();
```

```
                        go=true;
                      } // d.getRoot() == null
                }//d.getQuantifier != null
                else{
                   //error
                   d.setCurrent(start);
                   while( ii < newT.size()){
                while(j < Qvar.size()){
                   if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
                      ii=ii+1;
                   else{
                      if(j == Qvar.size() -1){
                         finish = (String)newT.elementAt(ii);
                         ii=ii+100;
                         j=j+100;
                       }
                      else j=j+1;
                   }
                }
             }
                   return finish;
                }
              }//go =true
             }//end not goingright

             }//else if Predicate

             else{
                d.getLeft();
                goingright=false;
             }
          System.out.println(pt);

          } // end while loop
      d.setCurrent(start);

          while( ii < newT.size()){
                 while(j < Qvar.size()){
                   if( newT.elementAt(ii).equals(Qvar.elementAt(jj)))
                      ii=ii+1;
                   else{
                      if(j == Qvar.size() -1){
                         finish = (String)newT.elementAt(ii);
                         ii=ii+100;
                         j=j+100;
                       }
                      else j=j+1;
                   }
                }
             }
      return finish;
     }

}// class CaptureTree


/*************************************************************
Name:   ReplaceCapture.java
Description:   Replace a variable in a binary tree in a Capture situation
Data: January 2006
*************************************************************/
package MS;


import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
```

```
import java.io.*;

public class ReplaceCapture{


  public BinaryTree ReplaceCapture( BinaryTree d, String a, String b) {

      ChildNode start = new ChildNode() ;
      ChildNode opRoot = new ChildNode();
      ChildNode current = new ChildNode();
      BinaryTree t = new BinaryTree() ;
      String pt = "" ;
      String vari ="";
      String quant = "";
      String predi = "";
      String oper = "";
      String pred="";
      String rootS="";
      String old = a ;
      String newv =b;
      boolean go = true;
      boolean crossroot=false;//did the program cross the root.
      boolean goingright=false;//did the program go to the right side of the tree.
      boolean opRootCheck=true;/* to help identify the operator at the top of the parse
      tree.*/
      boolean loop=true;
      boolean free=true;
      int goingrightcount=0;// how many times did the program go right.
      int goingrighthelp=0;
      int goingup = 0;//helps identify if the program is going up the parse tree.
      int startOP =0;/* identifies if the first item the program identifies is an
      operator.*/
      int QuantP = 0;// counts how many quantifiers so far.
      int count = 0 ;
      int count1OP = 0;//counts and keeps track of the operators.
      int count1Q = 0;/*identifies if the first item the program identifies is a
      quantifier.*/
      int count2Q = 0;//identifies how many quantifiers in while loop.
      int i = 0,jj =0,jjhelp=0;
      int stopLoop=15;

       start = d.getCurrent();
       rootS = d.getCurrent().getData();
       d.setCurrent(start);

       if(d.getQuantifier() != null) {
          if( d.getNot() == true) pt = pt + "\u00AC" +d.getData();
          else  pt = pt +d.getData();
          count1Q = count1Q + 1;

      if( a.compareTo(d.getVariable1())==0) {
          d.inVariable1(b);
          if("*" != d.getVariable2())
               d.inData(d.getQuantifier()+'('+b+','+d.getVariable2()+')');
           else
               d.inData(d.getQuantifier()+'('+b+')');
           }

       if( a.compareTo(d.getVariable2())==0) {
           d.inVariable2(b);
           if("*" != d.getVariable1())
               d.inData(d.getQuantifier()+'('+d.getVariable1()+','+b+')');
            else
               d.inData(d.getQuantifier()+'('+b+')');
            }
       }
       else if (d.getOperator() != null) {
```

```
    startOP =1;
    opRootCheck=false;
    opRoot=d.getCurrent();
    d.getLeft();
    count1OP = count1OP +1 ;
 }
 else if (d.getPredicate() != null) {
    i=i+14;
    if( a.compareTo(d.getVariable1())==0) {
        d.inVariable1(b);
        if("*" != d.getVariable2())
            d.inData(d.getPredicate()+'('+b+','+d.getVariable2()+')');
        else
            d.inData(d.getPredicate()+'('+b+')');
        }
        if( a.compareTo(d.getVariable2())==0) {
            d.inVariable2(b);
            if("*" != d.getVariable1())
                d.inData(d.getPredicate()+'('+d.getVariable1()+','+b+')');
            else
                d.inData(d.getPredicate()+'('+b+')');
        }
    d.setCurrent(start);
    //return d;
}
else {}

 while(i < stopLoop) {
   i = i +1 ;

   if(d.getCurrent().getQuantifier() != null){

       if( a.compareTo(d.getVariable1())==0) {
           d.inVariable1(b);
           if("*" != d.getVariable2())
             d.inData(d.getQuantifier()+'('+b+','+d.getVariable2()+')');
           else
             d.inData(d.getQuantifier()+'('+b+')');
          }
        if( a.compareTo(d.getVariable2())==0) {

           d.inVariable2(b);
           if("*" != d.getVariable1())
             d.inData(d.getQuantifier()+'('+d.getVariable1()+','+b+')');
           else
             d.inData(d.getQuantifier()+'('+b+')');
          }
       QuantP += QuantP;
       count2Q = count2Q +1 ;
    if(d.getQuantifier() == "0")
       return d;
    if(d.getCurrent().getRight() == null) {
       d.getLeft();
     }
     else {
      d.getRight() ;
     }
    }
   else if(d.getOperator() != null) {
    if(opRootCheck == true) {
       opRoot=d.getCurrent();
       opRootCheck = false;
     }
   if(goingup == 1) {
     pt=pt+'(';
     if( count1OP > 0) {
       while( count1OP != 0 ){
```

```
        d.getParent() ;
        count1OP = count1OP - 1 ;
       }// end while
      d.setCurrent(start);
      return d ;
     }

  }
else {
count1OP = count1OP + 1 ;
if(count1OP - 1 > 0)
  pt = pt+"(";
d.getLeft(); goingright=false;
  }
 } // end else goingup

else if(d.getPredicate() != null) {
     pred=d.getData();

        if( a.compareTo(d.getVariable1())==0) {
          d.inVariable1(b);
        if("*" != d.getVariable2())
          d.inData(d.getPredicate()+'('+b+','+d.getVariable2()+')');
        else
          d.inData(d.getPredicate()+'('+b+')');
       }

     if( a.compareTo(d.getVariable2())==0) {
        d.inVariable2(b);
        if("*" != d.getVariable1())
          d.inData(d.getPredicate()+'('+d.getVariable1()+','+b+')');
        else
            d.inData(d.getPredicate()+'('+b+')');
       }
 if(goingright==true){
   goingright = false;
   d.getParent();
   if(goingrighthelp == 4)
     jjhelp=2;
   else
     jjhelp=1;
     while( loop==true){
            if(d.getCurrent() == d.getRoot()) loop = false;
            if(d.getQuantifier() != null){
                d.getParent();
                if(d.getCurrent() == d.getRoot()) loop = false;
              }
            else if(d.getOperator() != null){
                   if(jj==jjhelp){
                     loop=false;
                    }
                    else {
                      d.getParent();
                       if(d.getCurrent() == d.getRoot()) loop =false;
                        jj=jj+1;
                    }
            }
            else{
              d.getParent();
              if(d.getCurrent() == d.getRoot()) loop =false;
            }
           }
     //}
      jj=jj-jj;
      loop=true;
      if(goingrighthelp == 4) goingrighthelp=0;
   go=true;
```

```
  while( go ==true){
   if( d.getOperator() != null){
      if(d.getCurrent() == d.getRoot() || d.getCurrent()==opRoot){
         if(crossroot == true){
            pt=pt+")";
            d.setCurrent(start);
            return d;
          } // if crossroot == true
         else {
            go=false;
            crossroot=true;
            pt=pt+")"+d.getData();
            d.getRight();
            goingright=false;

          }// crossroot==false
       }// d.getRoot != null
      else {
         go=false;
         goingright=true;
         goingrighthelp=goingrighthelp+1;
         goingrightcount=goingrightcount + 1;
         pt=pt+d.getData();
         d.getRight();

       } // d.getRoot == null
    }// if "d.getOperator()
   else if(d.getQuantifier() != null){
       if(d.getRoot() == d.getCurrent()){
          d.setCurrent(start);
          return d;
       }
       else{
          d.getParent();
       }//not root
   }//d.getQuantifier
   else {
      //error
      d.setCurrent(start);
      return d;
   }
  } // while "go" loop
}//end goingright
else {
  d.getParent();
  go=true;
  while(go ==true){
    if(d.getOperator() != null){

      go=true;
      while( go == true){
        if(d.getOperator() != null){
          if(goingrightcount >0){
            goingrightcount = goingrightcount - 1;
            }
          if(goingrightcount == 0){
              if(d.getCurrent() == d.getRoot()|| d.getCurrent()==opRoot){
               if(crossroot == false){
                 crossroot = true;
                 go = false;
                 pt=pt+d.getData();
                 d.getRight();
                }
               else {
                 d.setCurrent(start);
                 return d  ;                          }
                 }
```

```
                        }// d.getRoot() != null
                       else{
                         goingrighthelp=goingrighthelp+1;
                         goingrightcount = goingrightcount + 1;
                         goingright =true;
                         pt=pt+d.getData();
                         d.getRight();
                         go=false;
                         //return pt;
                        } //d.getRoot() == null
                     }// goingrightcount == 0
                    else{
                      //go=true;
                      //d.getParent();
                      goingrighthelp=goingrighthelp+2;  //+1
                         goingrightcount = goingrightcount + 1;
                         goingright =true;
                         pt=pt+d.getData();
                         d.getRight();
                         go=false;
                    }
                   }
                  }// go ==true
               }//d.getOperator !=null
               else if(d.getQuantifier() != null){
                       if(d.getRoot() == d.getCurrent() || d.getCurrent() == opRoot){
                          d.setCurrent(start);
                          return d;
                       }//d.getRoot() != null
                       else {
                          d.getParent();
                          go=true;
                       } // d.getRoot() == null
               }//d.getQuantifier != null
               else{

                  d.setCurrent(start);
                  return d;
               }
             }//go =true
           }//end not goingright

           }//else if Predicate

           else{
              d.getLeft();
              goingright=false;
           }

        } // end while loop
   d.setCurrent(start);
   return d ;

  } //

}// class ReplaceCapture


/*************************************************************
Name:   ReplaceVariable.java
Description:   Replace a variable in a binary tree
Data: January 2006
*************************************************************/

package MS;

import java.util.Vector;
```

```
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class ReplaceVariable{


  public BinaryTree ReplaceVariable( BinaryTree d,String a,String b) {


      ChildNode start = new ChildNode() ;
      BinaryTree t = new BinaryTree() ;
      String pt = "" ;
      String vari ="";
      String quant = "";
      String predi = "";
      String oper = "";
      String old = a ;
      String newv =b;
      int goingup = 0; //helps identify if the program is going up the parse tree.
      int count = 0;
      int count1OP = 0;//counts and keeps track of the operators.
      int count1Q = 0;/*identifies if the first item the program identifies is a
      quantifier.*/
      int count2Q = 0;//identifies how many quantifiers in while loop.
      int stopLoop=15;//helps prevent endless loops.

        start = d.getCurrent()
        d.setCurrent(start);
        d.pretrav();

        if(d.getQuantifier() != null) {

             if( a.equals(d.getCurrent().getVariable1())) {
               d.inVariable1(b);// \u2200 is unicode for For All
               if("\u2200".equals(d.getCurrent().getQuantifier())){
                  if("*"!=d.getVariable2())
                     d.inData("\u2200"+b+","+d.getCurrent().getVariable2());
                  else d.inData("\u2200"+b);
                }
               if("\u2203".equals(d.getQuantifier())) {// \u2200 is unicode for There
                  Exits
                  if("*"!=d.getVariable2())
                     d.inData("\u2203"+b+","+d.getCurrent().getVariable2());
                  else d.inData('\u2203'+b);
                }
               if("1".equals(d.getQuantifier())) {
                   if("*"!=d.getVariable2())
                     d.inData("1"+b+","+d.getCurrent().getVariable2());
                  else d.inData('1'+b);
                }
               }
              if( a.compareTo(d.getVariable2())==0) {

              d.inVariable2(b);
               if("\u2200".equals(d.getCurrent().getQuantifier())){
                  if("*"!=d.getVariable2())
                     d.inData("\u2200"+d.getCurrent().getVariable1()+","+b);
                  else d.inData("\u2200"+b);
                 }
               if("\u2203".equals(d.getCurrent().getQuantifier())) {
                  if("*"!=d.getVariable2())
                    d.inData("\u2203"+d.getCurrent().getVariable1()+","+b);
                  else d.inData('\u2203'+b);
                }
               if("1".equals(d.getCurrent().getQuantifier())) {
                  if("*"!=d.getVariable2())
```

```
                d.inData("1"+d.getCurrent().getVariable1()+","+b);
              else d.inData('1'+b);
             }
           }
      count1Q = count1Q + 1;
      d.getRight();

      while(d.getQuantifier() != null){
           if( a.equals(d.getVariable1())) {
             d.inVariable1(b);
             if("\u2200".equals(d.getQuantifier())){// \u2200 is unicode for For All
                if("*"!=d.getVariable2())
                   d.inData("\u2200"+b+","+d.getCurrent().getVariable2());
                else d.inData("\u2200"+b);
             }
             if("\u2203".equals(d.getQuantifier())){// \u2200 is unicode There
                  exists
                if("*"!=d.getVariable2())
                   d.inData("\u2203"+b+","+d.getCurrent().getVariable2());
                else d.inData('\u2203'+b);
             }
             if("1".equals(d.getQuantifier())){
                if("*"!=d.getVariable2())
                   d.inData("1"+b+","+d.getCurrent().getVariable2());
                else d.inData('1'+b);
              }
             }
             if( a.compareTo(d.getVariable2())==0) {
             d.inVariable2(b);
             if("\u2200".equals(d.getQuantifier())){// \u2200 is unicode for For All
               if("*"!=d.getVariable2())
                d.inData("\u2200"+d.getCurrent().getVariable1()+","+b);
               else d.inData("\u2200"+b);
             }
             if("\u2203".equals(d.getQuantifier())){// \u2200 is unicode for There
                 Exists
               if("*"!=d.getVariable2())
                d.inData("\u2203"+d.getCurrent().getVariable1()+","+b);
             else d.inData('\u2203'+b);
             }
             if("1".equals(d.getQuantifier())){
               if("*"!=d.getVariable2())
                d.inData("1"+d.getCurrent().getVariable1()+","+b);
                else  d.inData('1'+b);
               }
             }
           count1Q = count1Q + 1;
            if(d.getRight() != null) d.getRight();
            else d.getLeft();
          }
}
else if (d.getOperator() != null){
   d.getLeft();
   count1OP = count1OP +1 ;
}
else if (d.getPredicate() != null) {
    if( a.compareTo(d.getVariable1())==0){
             d.inVariable1(b);
             if("*" != d.getVariable2())
               d.inData(d.getPredicate()+'('+b+','+d.getVariable2()+')');
             else
               d.inData(d.getPredicate()+'('+b+')');
            }
          if( a.compareTo(d.getVariable2())==0) {
             d.inVariable2(b);

             if("*" != d.getVariable1())
```

```
            d.inData(d.getPredicate()+'('+d.getVariable1()+','+b+')');
          else
            d.inData(d.getPredicate()+'('+b+')');
        }
    i=i+14;
    d.setCurrent(start);
    return d;
}
else {}

 while(i <stopLoop) {
   i = i +1 ;
   if(d.getCurrent().getQuantifier() != null){
     if( a.compareTo(d.getVariable1())==0) {
          d.inVariable1(b);
          d.inVariable1(b);
          if("\u2200".equals(d.getQuantifier())){// \u2200 is unicode for For All
            if("*"!=d.getVariable2())
               d.inData("\u2200"+b+","+d.getCurrent().getVariable2());
            else d.inData("\u2200"+b);
          }
          if("\u2203".equals(d.getQuantifier())){// \u2200 is unicode for There
              Exists
             if("*"!=d.getVariable2())
                d.inData("\u2203"+b+","+d.getCurrent().getVariable2());
             else d.inData('\u2203'+b);
          }
          if("1".equals(d.getQuantifier())){
            if("*"!=d.getVariable2())
               d.inData("1"+b+","+d.getCurrent().getVariable2());
            else d.inData('1'+b);
           }
         }
      if( a.compareTo(d.getVariable2())==0) {
          d.inVariable2(b);
          d.inVariable2(b);
          if("\u2200".equals(d.getQuantifier())) {
            if("*"!=d.getVariable2())
              d.inData("\u2200"+d.getCurrent().getVariable1()+","+b);
            else d.inData("\u2200"+b);
          }
          if("\u2203".equals(d.getQuantifier())) {
            if("*"!=d.getVariable2())
              d.inData("\u2203"+d.getCurrent().getVariable1()+","+b);
            else d.inData('\u2203'+b);
          }
          if("1".equals(d.getQuantifier())) {
             if("*"!=d.getVariable2())
               d.inData("\u2200"+d.getCurrent().getVariable1()+","+b);
             else d.inData('1'+b);
           }
          }
    count2Q = count2Q +1 ;
    if(d.getCurrent().getRight() == null) {
       d.getLeft();
     }
     else {
      d.getRight() ;
      }
    }
   else if(d.getOperator() != null) {
   if(goingup == 1) {
     if( count1OP > 0) {
       while( count1OP != 0 ){
          d.getParent() ;
          count1OP = count1OP - 1 ;
        }// end while
```

```
      d.setCurrent(start);
      return d ;
     }
   }
else {
count1OP = count1OP + 1 ;
d.getLeft();
if(d.getCurrent().getQuantifier() != null)
   count2Q = count2Q + 1
  }
 } // end else goingup
else if(d.getPredicate() != null){
  if( a.equals(d.getCurrent().getVariable1())) {
        d.inVariable1(b);
        d.inVariable1(b);
        if("*" != d.getVariable2())
          d.inData(d.getPredicate()+'('+b+','+d.getVariable2()+')');
        else
          d.inData(d.getPredicate()+'('+b+')');
       }
     if( a.equals(d.getCurrent().getVariable2())) {
        d.inVariable2(b);
        if("*" != d.getVariable1())
           d.inData(d.getPredicate()+'('+d.getVariable1()+','+b+')');
        else
           d.inData(d.getPredicate()+'('+b+')');
       }
  d.pretrav();
  d.getParent() ;
  if(count1OP < 1) {
     d.setCurrent(start);
     return d;
   }
     if(count2Q == 0) {
       if(count1OP == 0 ) {
        if(count1Q > 0) {
            while( count1Q > 0) {
            d.getParent();
            count1Q= count1Q -1;
              if( start == d.getCurrent() ){d.setCurrent(start);  return d; }
            }
          }
         }
     }
      if(d.getCurrent().getOperator() != null) {
       count1OP = count1OP - 1 ;
       d.getRight();
         if(d.getPredicate() != null) {
           if( a.compareTo(d.getVariable1())==0) {
              d.inVariable1(b);
              d.inVariable1(b);
              d.inData(d.getPredicate()+'('+b+')');
            }
           if( a.compareTo(d.getVariable2())==0) {
              d.inVariable2(b);
              d.inVariable2(b);
              d.inData(d.getPredicate()+'('+b+')');
            }
          d.getParent() ;
          if(d.getCurrent() == start){
             d.setCurrent(start);
             return d;
             }
             goingup = goingup + 1 ;
             if(d.getCurrent().getParent() != null) {
               d.getParent() ;
                if(d.getCurrent().getQuantifier() != null) {
```

```
                        if(d.getCurrent() == start) {
                          d.setCurrent(start);
                          return d;
                          }
                        while( d.getCurrent().getQuantifier() != null) {
                          d.getParent() ;
                         }
                      }
                    } // end if getParen
                    if(start == d.getCurrent()) {
                       if( count1OP < 1){
                         d.setCurrent(start);
                         return d;
                        }
                    }
                    if(d.getOperator() != null){
                        if( count1OP > 0) {
                          count1OP = count1OP -1 ;
                          d.getRight() ;
                          goingup = goingup - goingup ;
                         } // end if count1op
                         else {
                           if( count1Q != 0 ) {
                             while ( count1Q != 0) {
                               d.getParent() ;
                               if( d.getCurrent() == start) {
                                 d.setCurrent(start);
                                 return d ;
                                }
                               count1Q = count1Q - 1 ;
                             } // end while
                           } // end if

                        } // end else

                    } // end if d.getOperator ! = null
                    if(d.getPredicate() != null) {
                        d.setCurrent(start);
                        return d;
                      }
                   //d.getParent();
                  }//end if d.Predicate
              }// end if d.getOperator
          }//else if Predicate
       } // end while loop
   d.setCurrent(start);
   return d ;
  } //

}// class ReplaceVariable


/*************************************************************
Name:   ParseTree.java
Description:   Parse a WFF into a String.
Data: January 2005
*************************************************************/

package MS;


import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class ParseTree{
```

```
public String ParseTree( BinaryTree d) {

    ChildNode start = new ChildNode() ;
    ChildNode opRoot = new ChildNode();
    BinaryTree t = new BinaryTree() ;
    String pt = "" ;
    String vari ="";
    String quant = "";
    String predi = "";
    String oper = "";
    String pred="";
    String rootS="";
    boolean go = true;
    boolean crossroot=false;//did the program cross the root.
    boolean goingright=false;//did the program go to the right side of the tree.
    boolean opRootCheck=true;/* to help identify the operator at the top of the parse
    tree.*/
    boolean loop=true;
    int goingrightcount=0;// how many times did the program go right.
    int goingrighthelp=0;
    int goingup = 0;//helps identify if the program is going up the parse tree.
    int startOP =0;/* identifies if the first item the program identifies is an
    operator.*/
    int QuantP = 0;// counts how many quantifiers so far.
    int count = 0 ;
    int count1OP = 0;//counts and keeps track of the operators.
    int count1Q = 0;/*identifies if the first item the program identifies is a
    quantifier.*/
    int count2Q = 0;//identifies how many quantifiers in while loop.
    int i = 0 ,jj =0,jjhelp=0;
    int stopLoop=15;//helps prevent endless loops.
    start = d.getCurrent();
    rootS = d.getCurrent().getData();

    System.out.println("********^^ParseJava^^^^^^^^^^^^^^");

    d.setCurrent(start);

      if(d.getQuantifier() != null) { //\u00AC is unicode for negation
        if( d.getNot() == true) pt = pt + "\u00AC" +d.getData();
        else  pt = pt +d.getData();
        count1Q = count1Q + 1;

        if(d.getCurrent().getRight() != null) d.getRight();
        else d.getLeft();
        while(d.getQuantifier() != null){  //\u00AC is unicode for negation
            if( d.getNot() == true) pt = pt + "\u00AC" +d.getData();
            else  pt = pt + d.getData();
            count1Q = count1Q + 1;
            d.getRight();
         }
         QuantP += QuantP;
         //pt=pt+"(";
      }
      else if (d.getOperator() != null) {//\u00AC is unicode for negation
         startOP =1;
        if( d.getNot() == true) pt = pt + "\u00AC"+"(";
        opRootCheck=false;
        opRoot=d.getCurrent();
        d.getLeft();
        count1OP = count1OP +1 ;
      }
      else if (d.getPredicate() != null) {
         if( d.getNot() == true) pt = pt + "\u00AC" +d.getData();
         else  pt = pt + d.getData();
```

```
    i=i+14;
    d.setCurrent(start);
    return pt;
}
else {}

 while(i < stopLoop) {
  i = i +1 ;
  if(d.getCurrent().getQuantifier() != null){ //\u00AC is unicode for negation
     if( d.getNot() == true)
         pt = pt + "\u00AC"+ d.getData() ; //(
     else pt = pt + d.getData() ; //(
     QuantP += QuantP;
     count2Q = count2Q +1 ;
    if(d.getQuantifier() == "0") return pt;
    if(d.getCurrent().getRight() == null) {
        System.out.println("getting left") ;
        d.getLeft();
     }
     else d.getRight() ;
    }
    else if(d.getOperator() != null) {

    if(opRootCheck == true) {
       opRoot=d.getCurrent();
       opRootCheck = false;
     }

   if(goingup == 1) {
     pt=pt+'(';
     if( count1OP > 0) {
       while( count1OP != 0 ){
          d.getParent() ;
          count1OP = count1OP - 1 ;
         }// end while
        d.setCurrent(start);
        return pt ;
       }
     }
   else {//\u00AC is unicode for negation
   if( d.getNot() == true) pt = pt + "\u00AC"; //(
   else  pt = pt; //(
   count1OP = count1OP + 1 ;
   if(count1OP - 1 > 0)
     pt = pt+"(";
   d.getLeft(); goingright=false;
     }
    } // end else goingup

   else if(d.getPredicate() != null) {
        pred=d.getData();
        if(d.getOperator() != null){//\u00AC is unicode for negation
             if( d.getNot() == true)
                pt  = pt +"("+"\u00AC" +pred; //( )
             else  pt = pt +"("+ pred; //( )
         }
         else {//\u00AC is unicode for negation
            if( d.getNot() == true)
               pt = pt +"\u00AC" + pred; //( )
            else  pt = pt + pred; //( )
          }
    if(goingright==true){
      goingright = false;
      d.getParent();
      if(goingrighthelp == 4)
        jjhelp=2;
      else
```

```
  jjhelp=1;
   while( loop==true){
          if(d.getCurrent() == d.getRoot()) loop = false;
          if(d.getQuantifier() != null){
              d.getParent();
              if(d.getCurrent() == d.getRoot()) loop = false;
            }
          else if(d.getOperator() != null){
                  if(jj==jjhelp){
                    loop=false;
                   }
                   else {
                     d.getParent();
                      if(d.getCurrent() == d.getRoot()) loop =false;
                      jj=jj+1;
                   }
          }
          else{
            d.getParent();
            if(d.getCurrent() == d.getRoot()) loop =false;
          }
         }
  //}
   jj=jj-jj;
   loop=true;
   if(goingrighthelp == 4)
     goingrighthelp=0;
go=true;
while( go ==true){
 if( d.getOperator() != null){
    if(d.getCurrent() == d.getRoot() || d.getCurrent()==opRoot){
       if(crossroot == true){
          pt=pt+")";
          d.setCurrent(start);
          return pt;
        } // if crossroot == true
       else {
          go=false;
          crossroot=true;
          pt=pt+")"+d.getData();
          d.getRight();
          goingright=false;
        }// crossroot==false
     }// d.getRoot != null
    else {
       go=false;
       goingright=true;
       goingrighthelp=goingrighthelp+1;
       goingrightcount=goingrightcount + 1;
       pt=pt+d.getData();
       d.getRight();

     } // d.getRoot == null
  }// if "d.getOperator()
 else if(d.getQuantifier() != null){
     if(d.getRoot() == d.getCurrent()){
        d.setCurrent(start);
        return pt;
     }
     else{
        d.getParent();
    }//not root
 }//d.getQuantifier
 else {

    d.setCurrent(start);
    return pt;
```

```
   }
  } // while "go" loop
}//end goingright
else {
  d.getParent();
  go=true;
  while(go ==true){
    if(d.getOperator() != null){
      go=true;
      while( go == true){
        if(d.getOperator() != null){
          if(goingrightcount >0){
            goingrightcount = goingrightcount - 1;
            }
          if(goingrightcount == 0){

              if(d.getCurrent() == d.getRoot()|| d.getCurrent()==opRoot){
               if(crossroot == false){
                  crossroot = true;
                  go = false;
                  pt=pt+d.getData();
                  d.getRight();
                }
               else {
                  d.setCurrent(start);
                  return pt;
               }

             }// d.getRoot() != null
            else{
              goingrighthelp=goingrighthelp+1;
              goingrightcount = goingrightcount + 1;
              goingright =true;
              pt=pt+d.getData();
              d.getRight();
              go=false;
              //return pt;
             } //d.getRoot() == null
          }// goingrightcount == 0
         else{
           go=true;
           d.getParent();
         }
        }
      }// go ==true
    }//d.getOperator !=null
    else if(d.getQuantifier() != null){
            if(d.getRoot() == d.getCurrent() || d.getCurrent() == opRoot){
               d.setCurrent(start);
               return pt;
            }//d.getRoot() != null
            else {
               d.getParent();
               go=true;S
            } // d.getRoot() == null
    }//d.getQuantifier != null
    else{
       //error
       d.setCurrent(start);
       return pt;
    }
  }//go =true
}//end not goingright

}//else if Predicate

else{
```

```
            d.getLeft();
            goingright=false;
          }
        } // end while loop

   d.setCurrent(start);
   return pt ;
  } //

}// class ParseTree


/*************************************************************
Name:   SearchTree.java
Description:   Search a variable in a binary tree in a capture situation
Data: January 2006
*************************************************************/

package MS;


import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class SearchTree{


  public String SearchTree( BinaryTree d,String s) {

      ChildNode start = new ChildNode() ;
      ChildNode opRoot = new ChildNode();
      BinaryTree t = new BinaryTree() ;
      Vector newT = new Vector() ;
      String pt="";
      String vari ="";
      String quant = "";
      String predi = "";
      String oper = "";
      String pred="";
      String rootS="";
      String Svar1=s;
      String Svar2=s;
      String var1="";
      String var2="";
      boolean go = true;
      boolean crossroot=false;//did the program cross the root.
      boolean goingright=false;//did the program go to the right side of the tree.
      boolean opRootCheck=true;;/* to help identify the operator at the top of the parse
      tree.*/
      boolean loop=true;
      int goingrightcount=0;// how many times did the program go right
      int goingrighthelp=0;
      int goingup = 0;//helps identify if the program is going up the parse tree.
      int startOP =0;/* identifies if the first item the program identifies is an
      operator.
      int QuantP = 0;// counts how many quantifiers so far.
      int count = 0 ;
      int count1OP = 0;//counts and keeps track of the operators.
      int count1Q = 0;/*identifies if the first item the program identifies is a
      quantifier.*/
      int count2Q = 0;//identifies how many quantifiers in while loop.
      int i = 0 ;
      int jj =0;
      int jjhelp=0;
      int stopLoop=15;//helps prevent endless loops.
```

```
start = d.getCurrent();
rootS = d.getCurrent().getData();

System.out.println("**************SearchTree*******");

d.setCurrent(start);

   count1Q = count1Q + 1;

   if(d.getCurrent().getRight() != null) d.getRight();
   else d.getLeft();

   while(d.getQuantifier() != null){
         count1Q = count1Q + 1;
         var1=d.getVariable1();
         var2=d.getVariable2();
            d.getRight();
    }
    QuantP += QuantP;

 }
 else if (d.getOperator() != null) {
   startOP =1;
   opRootCheck=false;
   opRoot=d.getCurrent();
   d.getLeft();
   count1OP = count1OP +1 ;
 }

 else if (d.getPredicate() != null) {
    i=i+14;
    d.setCurrent(start);
    return pt;
}
else {}

 while(i < stopLoop) {
   i = i +1 ;
   if(d.getCurrent().getQuantifier() != null){
     QuantP += QuantP;
     count2Q = count2Q +1 ;

    if(d.getQuantifier() == "0")
       return "ok";

    if(d.getCurrent().getRight() == null) {
        System.out.println("getting left") ;
       var1=d.getVariable1();System.out.println(d.getVariable1()+"1"+Svar2);
         var2=d.getVariable2();System.out.println(d.getVariable2()+"2"+Svar2);
         if(Svar1.equals(var1))
            return Svar1;
         else if(Svar1.equals(var2))
            return Svar1;
         else if(Svar2.equals(var1))
            return Svar2;
         else if(Svar2.equals(var2))
            return Svar2;
         else
       d.getLeft();

     }
     else {
       var1=d.getVariable1(); System.out.println(d.getVariable1()+"1");
         var2=d.getVariable2(); System.out.println(d.getVariable2()+"2");
         if(Svar1.equals(var1))
            return  Svar1;
```

```
      else if(Svar1.equals(var2))
         return  Svar1;
      else if(Svar2.equals(var1))
         return  Svar2;
      else if(Svar2.equals(var2))
         return  Svar2;
      else
         d.getRight();

  }

 }

else if(d.getOperator() != null) {

 if(opRootCheck == true) {
    opRoot=d.getCurrent();
    opRootCheck = false;
    System.out.println("opRoot");
  }

if(goingup == 1) {
  if( count1OP > 0) {
    while( count1OP != 0 ){
       d.getParent() ;
       count1OP = count1OP - 1 ;
      }// end while
     d.setCurrent(start);
     return "ok" ;
    }
  }
else {
count1OP = count1OP + 1 ;
d.getLeft(); goingright=false;
  }
 } // end else goingup

else if(d.getPredicate() != null) {
   pred=d.getData();

 if(goingright==true){
   goingright = false;
   //goingup= true;
   d.getParent();

   if(goingrighthelp == 4)
     jjhelp=2;
   else
     jjhelp=1;
    //if(d.getQuantifier() !=null){
      while( loop==true){
             if(d.getCurrent() == d.getRoot()) loop = false;
             //d.getParent();
             if(d.getQuantifier() != null){
                 d.getParent();
                 if(d.getCurrent() == d.getRoot()) loop = false;
               }
             else if(d.getOperator() != null){
                     if(jj==jjhelp){
                       loop=false;
                      }
                      else {
                        d.getParent();
                         if(d.getCurrent() == d.getRoot()) loop =false;
                         jj=jj+1;

                      }
```

```
            }
            else{
              d.getParent();
              if(d.getCurrent() == d.getRoot()) loop =false;
            }
           }
     //}
      jj=jj-jj;
      loop=true;

      if(goingrighthelp == 4)
        goingrighthelp=0;

   go=true;
   while( go ==true){
    if( d.getOperator() != null){
       if(d.getCurrent() == d.getRoot() || d.getCurrent()==opRoot){
          if(crossroot == true){
             d.setCurrent(start);
             return "ok";
           } // if crossroot == true
          else {
             go=false;
             crossroot=true;
             d.getRight();
             goingright=false;

           }// crossroot==false
        }// d.getRoot != null
       else {
          go=false;
          goingright=true;
          goingrighthelp=goingrighthelp+1;
          goingrightcount=goingrightcount + 1;
          d.getRight();

        } // d.getRoot == null
     }// if "d.getOperator()
    else if(d.getQuantifier() != null){
        if(d.getRoot() == d.getCurrent()){
           d.setCurrent(start);
           return "ok";
        }
        else{
           d.getParent();
        }//not root
    }//d.getQuantifier
    else {
       //error
       d.setCurrent(start);
       return pt;
    }
   } // while "go" loop
}//end goingright
else {
  d.getParent();


  go=true;
  while(go ==true){
    if(d.getOperator() != null){
      go=true;
      while( go == true){
        if(d.getOperator() != null){
          if(goingrightcount >0){
            goingrightcount = goingrightcount - 1;
```

```
                    }
                  if(goingrightcount == 0){

                      if(d.getCurrent() == d.getRoot()|| d.getCurrent()==opRoot){
                       if(crossroot == false){
                         crossroot = true;
                         go = false;

                        }
                       else {
                         d.setCurrent(start);
                         return "ok";
                       }

                      }// d.getRoot() != null
                     else{
                       goingrighthelp=goingrighthelp+1;
                       goingrightcount = goingrightcount + 1;
                       goingright =true;
                       d.getRight();
                       go=false;
                      } //d.getRoot() == null
                   }// goingrightcount == 0
                  else{
                    go=true;
                    d.getParent();
                  }
                 }
                }// go ==true
              }//d.getOperator !=null
              else if(d.getQuantifier() != null){
                    if(d.getRoot() == d.getCurrent() || d.getCurrent() == opRoot){
                       d.setCurrent(start);
                       return "ok";
                    }//d.getRoot() != null
                    else {
                       d.getParent();
                       go=true;
                    } // d.getRoot() == null
              }//d.getQuantifier != null
              else{

                 d.setCurrent(start);
                 return "ok";
              }
            }//go =true
          }//end not goingright

          }//else if Predicate

          else{

             d.getLeft();
             goingright=false;

          }

        } // end while loop

   d.setCurrent(start);
   return "ok" ;


  } //


}// class SearchTree
```

```
/*************************************************************
Name:   Sending.java
Description:   Send data from Interface.java to Output.java
Data: January 2006
*************************************************************/
package MS;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Sending {

 private static String a1="";
 private static String a2="";
 private static String c1="";


 public static void setA1(String a){
      a1 = a ;
 }
public static void setA2(String a){
      a2 = a ;
 }
public static void setC1(String a){
      c1 = a ;
 }

public static String getA1(){
      return a1;
 }
public static String getA2(){
      return a2;
 }
public static String getC1(){
      return c1;
 }


}//class Sending


/*************************************************************
Name:   STree.java
Description:   Create a Semantic Tableaux tree.
Data: January 2006
*************************************************************/
package MS;

import java.util.Vector;
import java.lang.Object;
import java.util.StringTokenizer;
import java.io.*;

public class STree{
        private ChildNode rroot,p ;
        private ChildNode Child ;
        private ChildNode right,left ;
        private BinaryTree temp;
        String lleft, rright;

        public BinaryTree STree(BinaryTree newtree, BinaryTree d, String s, String r) {
```

```
String A ="";
String A1="";
String newV1 = "";
String newV2 = "";
String oldV ="";
String oneRight="";
String oneLeft="";
String nodeleft="";
String noderight="";
String rVar1= d.getVariable1();
String rVar2 = d.getVariable2();
Vector vA = new Vector();
Vector one1 = new Vector();
Vector two2 = new Vector();

ChildNode home = new ChildNode() ;
ChildNode rootA = new ChildNode() ;
ChildNode rootB = new ChildNode();
ChildNode rootA1 = new ChildNode() ;
ChildNode rootA2 = new ChildNode() ;
ChildNode rootA3 = new ChildNode();
ChildNode rootAA = new ChildNode() ;
ChildNode rootRAA = new ChildNode() ;
ChildNode rootLAA = new ChildNode() ;
ChildNode rootRBB = new ChildNode() ;
ChildNode rootLBB = new ChildNode() ;
ChildNode rootBB = new ChildNode() ;
ChildNode rootA11 = new ChildNode() ;
ChildNode rootA22 = new ChildNode() ;
ChildNode rootOneTree1 = new ChildNode() ;
ChildNode rootOneTree2 = new ChildNode() ;
ChildNode QTreeNode = new ChildNode(); // used in '1' quantifier.
ChildNode tempBranch = new ChildNode();

BinaryTree treeA= new BinaryTree() ;
BinaryTree treeA1= new BinaryTree() ;
BinaryTree treeA2= new BinaryTree() ;
BinaryTree treeA3= new BinaryTree();
BinaryTree treeAA= new BinaryTree() ; // used in '1' quantifier.
BinaryTree treeRAA= new BinaryTree() ;
BinaryTree treeLAA= new BinaryTree() ;
BinaryTree treeA11= new BinaryTree() ; // used in '1' quantifier.
BinaryTree treeA22= new BinaryTree() ; // used in '1' quantifier.
BinaryTree treeB= new BinaryTree() ;
BinaryTree treeBB= new BinaryTree() ;
BinaryTree treeRBB= new BinaryTree() ;
BinaryTree treeLBB= new BinaryTree() ;
BinaryTree onetree1 = new BinaryTree();
BinaryTree onetree2 = new BinaryTree();
BinaryTree QTree = new BinaryTree(); //  used in '1' quantifier.
Pa parse = new Pa();
ParseTree pt = new ParseTree();
BuildTree bt = new BuildTree();
BuildTree onet = new BuildTree();
ReplaceVariable rv = new ReplaceVariable();
CopyTree treecopy= new CopyTree();
String tempVar, newVar ;
newV1 =s;
newV2 =r;
treeA.setRoot(rootA);
treeA1.setRoot(rootA1);
treeA2.setRoot(rootA2);
treeA3.setRoot(rootA3);
treeAA.setRoot(rootAA);
treeRAA.setRoot(rootRAA);
treeLAA.setRoot(rootLAA);
```

```
treeA11.setRoot(rootA11);
treeA22.setRoot(rootA22);
treeB.setRoot(rootB);
treeBB.setRoot(rootBB);
treeRBB.setRoot(rootRBB);
treeLBB.setRoot(rootLBB);
onetree1.setRoot(rootOneTree1);
onetree2.setRoot(rootOneTree2);
QTree.setRoot(QTreeNode);
home=d.getCurrent();
String oldVar1 = d.getVariable1();
String oldVar2 = d.getVariable2();

if( d.getCurrent().getQuantifier() != null) {
      if(d.getNot() == false) { // \u2200 is unicode for For All
        if("\u2200".equals(d.getQuantifier())) {
        if( newtree.getCenter() == false) newtree.setCenter();
        NewTreeSetRight(newtree);
        newtree.pretrav();
        treeA=d; treeA.getRight();
        treeAA=treecopy.CopyTree(treeA,treeAA);
        treeAA.setCurrent(treeAA.getRoot());
        newtree.pretrav();
        try{
        treeA=d;

        if(Character.isLowerCase(s.charAt(0))==true)
           treeAA=rv.ReplaceVariable(treeAA,rVar1 ,newV1);
        if(Character.isLowerCase(r.charAt(0))==true )
           treeAA=rv.ReplaceVariable(treeAA,rVar2 ,newV2);
        treeAA.setCurrent(treeAA.getRoot());
        if("*" != rVar2)
           treeAA=rv.ReplaceVariable(treeAA,rVar2, newV2);
        treeAA.setCurrent(treeAA.getRoot());
        A1=pt.ParseTree(treeAA);
        }
        catch(Exception e) {
        }
        newtree.inData(A1) ;
        newtree.inDataTree(treeAA);
        newtree.inNodeTree(treeAA.getCurrent().getData());
        inInfo(d,newtree);
        newtree.pretrav();
        newtree.pretrav();return newtree ;
       }//end for all
       else if ("\u2203".equals(d.getQuantifier())) {/* \u2200 is unicode for
             There Exist*/
        if( newtree.getCenter() == false) newtree.setCenter();
           treeA=d; treeA.getRight();
           NewTreeSetRight(newtree);

        try{
        treeA=d ;
        treeA.setRoot(treeA.getCurrent());

        treeAA=treecopy.CopyTree(treeA,treeAA);
        treeAA.setCurrent(treeAA.getRoot());

        if(Character.isLowerCase(s.charAt(0))==true)
           treeAA=rv.ReplaceVariable(treeAA,rVar1 ,newV1);
        treeAA.setCurrent(treeAA.getRoot());
        if(Character.isLowerCase(r.charAt(0))==true )
           treeAA=rv.ReplaceVariable(treeAA,rVar2 ,newV2);

        A1=pt.ParseTree(treeAA);
        }
        catch(Exception e) {
```

```
}
newtree.inNodeTree(treeAA.getCurrent().getData());
newtree.inData(A1) ;
newtree.inDataTree(treeAA);newtree.pretrav();
inInfo(d,newtree);
} //There exits
else  //(d.getCurrent().getQuantifier() == "1")
{
    d.getRight();
    treeA=treecopy.CopyTree(d,treeA);
    treeA.setCurrent(treeA.getRoot());
    treeA2=treecopy.CopyTree(treeA,treeA2);
    treeA1=rv.ReplaceVariable(treeA,rVar1,newV1);
    treeA.setCurrent(treeA.getRoot());

    QTree.setCurrent(QTree.getRoot());
    QTree.inData("\u2227"); // in 'and' operator
    QTree.inOperator("\u2227");
    QTree.insertRightGo();
    QTree.inData("\u2200"+rVar1); // in 'For All' operator
    QTree.inQuantifier("\u2200");
    QTree.inVariable1(rVar1);
    //QTree.inVariable1(rVar);
    QTree.insertRightGo();
    QTree.inData("\u2192"); // in 'if then operator'
    QTree.inOperator("\u2192");
    //QTree.inVariable1(rVar1);
    QTree.insertRightGo();
    QTree.inQuantifier("0");
    QTree.inData(rVar1 + "=" + newV1);
    QTree.inVariable1(rVar1 + "=" + newV1);
    QTree.getParent();
    QTree.insertLeftGo();
    QTree=treecopy.CopyTree(treeA2,QTree);
    QTree.setCurrent(QTree.getRoot());
    QTree.setCurrent(QTree.getRoot());
    QTree.insertLeftGo();
    QTree=treecopy.CopyTree(treeA1,QTree);
    QTree.setCurrent(QTree.getRoot());
    if( newtree.getCenter() == false) newtree.setCenter();
    QTree.setCurrent(QTree.getRoot());
    treeA1.setRoot(treeA1.getCurrent());
    treeA2.setRoot(treeA2.getCurrent());
    treeA.setRoot(treeA.getCurrent());
    treeAA.setRoot(treeAA.getCurrent());
    QTree.setCurrent(QTree.getRoot());
    treeA.setCurrent(treeA.getRoot());
    treeAA.setCurrent(treeAA.getRoot());

  NewTreeSetRight(newtree);
  QTree.getLeft();
  newtree.inLeftTree(QTree.getCurrent().getData());
  lleft = pt.ParseTree(QTree);System.out.println("4");
  QTree.setCurrent(QTree.getRoot());
  QTree.getRight();
  newtree.inRightTree(QTree.getCurrent().getData());
  rright =pt.ParseTree(QTree);
  QTree.setCurrent(QTree.getRoot());
  newtree.inDataTree(QTree);
  treeRAA=treecopy.CopyTree(QTree,treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.inRightDataTree(treeRAA);
  treeAA.setCurrent(treeAA.getRoot());
  QTree.setCurrent(QTree.getRoot());
  treeLAA=treecopy.CopyTree(QTree,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
```

```
                    treeLAA.setCurrent(treeLAA.getRoot());
                    newtree.inLeftDataTree(tree);
                    treeAA.setCurrent(treeAA.getRoot());
                    newtree.inOperator("\u2194");// in 'if and only if' operator
                    newtree.inLeftData(lleft);System.out.println("41 lleft, "+lleft);
                    newtree.inRightData(rright);System.out.println("42 rright,
                    "+rright);
                     //*******************************************************
                }
               } // end temp.getNot == false
//*****************************************************

               else {
                  if("\u2203".equals(d.getQuantifier()) ) {//\u2203='There Exists'
                     if( newtree.getCenter() == false) newtree.setCenter();
                     NewTreeSetRight(newtree);
                  try{
                  treeA=d ;
                  treeA.getRight();
                  treeAA=treecopy.CopyTree(treeA,treeAA);
                  treeAA.getCurrent().setNot();
                  treeAA.pretrav();
                  treeA.setCurrent(treeA.getRoot());
                  treeAA.setCurrent(treeAA.getRoot());

                  if(Character.isLowerCase(s.charAt(0))==true)
                     treeA=rv.ReplaceVariable(treeA,rVar1 ,newV1);
                  treeAA.setCurrent(treeAA.getRoot());
                  if(Character.isLowerCase(s.charAt(0))==true)
                     treeAA=rv.ReplaceVariable(treeAA,rVar1 ,newV1);
                  if(Character.isLowerCase(r.charAt(0))==true )
                      treeAA=rv.ReplaceVariable(treeAA,rVar2 ,newV2);
                  treeAA.setCurrent(treeAA.getRoot());
                  A1=pt.ParseTree(treeAA);
                  }
                  catch(Exception e) {
                  System.out.println("Failed ");
                  }
                  treeAA.setCurrent(treeAA.getRoot());
                  newtree.inQuantifier("1");
                  newtree.setNot();
                  treeA1.setCurrent(treeA1.getRoot());
                  treeA1.setNot();
                  newtree.inData(A1) ; newtree.pretrav();
                  newtree.inDataTree(treeAA);
                  newtree.inNodeTree(treeAA.getCurrent().getData());
                  inInfo(d,newtree);

                  } //There exits

                  else if ("\u2200".equals(d.getQuantifier())) {// \u2200='For All'
                  if( newtree.getCenter() == false) newtree.setCenter();
                     NewTreeSetRight(newtree);
                  try{
                  treeA=d ;
                  treeA.getRight();
                  treeAA=treecopy.CopyTree(treeA,treeAA);treeAA.getCurrent().setNot();
                  treeAA.setCurrent(treeAA.getRoot());
                  treeA.setCurrent(treeA.getRoot());
                  //System.out.println("Step one ");
                  if(Character.isLowerCase(s.charAt(0))==true)
                     treeAA=rv.ReplaceVariable(treeAA,rVar1 ,newV1);
                  treeAA.setCurrent(treeAA.getRoot());
                  if( Character.isLowerCase(r.charAt(0))==true)
```

```
    treeAA=rv.ReplaceVariable(treeAA,rVar2 ,newV2);
treeAA.setCurrent(treeAA.getRoot());
A1=pt.ParseTree(treeAA);
}
catch(Exception e) {
System.out.println("Failed ");
}
newtree.setNot();
newtree.inDataTree(treeAA);
newtree.inData(A1) ;
newtree.inNodeTree(treeAA.getCurrent().getData());
inInfo(d,newtree);
}
else  //(d.getQuantifier() == "1")
{
    d.getRight();
    treeA=d;

    treeA1=treecopy.CopyTree(treeA,treeA1);
    treeA1.setCurrent(treeA1.getRoot());
    treeA=d;

    treeA3=treecopy.CopyTree(treeA,treeA3);
    treeA1.insertParentGo();
    treeA1.setRoot(treeA1.getCurrent());
    treeA1.inData("\u2203"+rVar1+","+newV1);//in 'there exist'
    treeA1.inQuantifier("\u2203");
    treeA1.inVariable1(rVar1); treeA=d;
    treeA2=treecopy.CopyTree(treeA,treeA2);

    treeA2=rv.ReplaceVariable(treeA2,rVar1,newV1);
    treeA2.setCurrent(treeA2.getRoot());
    treeA3.insertParentGo(); treeA3.setRoot(treeA3.getCurrent());
    treeA3.inData("\u2228"); // \u2228='or' operator
    treeA3.inOperator("\u2228");
    treeA3.insertRightGo();System.out.println("553
    ");treeA3.pretrav();
    treeA3=treecopy.CopyTree(treeA2,treeA3);
    treeA3.setCurrent(treeA3.getRoot());
    treeA3.insertParentGo();

    treeA3.setRoot(treeA3.getCurrent());
    treeA3.inData("\u2192"); // \u2192='if then' operator
    treeA3.inOperator("\u2192");
    treeA3.insertRightGo();
    treeA3.inData(rVar1+"="+newV1);System.out.println("554
    ");treeA3.pretrav();
    treeA3.inQuantifier("0");
    treeA3.inVariable1(rVar1+"="+newV1);
    treeA3.setCurrent(treeA3.getRoot());

    treeA3.insertParentGo(); treeA3.setRoot(treeA3.getCurrent());
    treeA3.inData("\u2203"+rVar1); // \u2203='There Exists"
    treeA3.inQuantifier("\u2203"); treeA3.inVariable1(rVar1);
    treeA3.setCurrent(treeA3.getRoot());
    QTree.setCurrent(QTree.getRoot());
    QTree.inData("\u2192");// \u2192='if then' operator
    QTree.inOperator("\u2192");
    QTree.insertRightGo();
    QTree.pretrav();
    QTree=treecopy.CopyTree(treeA3,QTree);
    QTree.pretrav();
    QTree.setCurrent(QTree.getRoot());
    QTree.insertLeftGo();
    QTree=treecopy.CopyTree(treeA1,QTree);
    QTree.setCurrent(QTree.getRoot());
```

```
                if( newtree.getCenter() == true) newtree.setCenter();
                QTree.setCurrent(QTree.getRoot());
                QTree.setCurrent(QTree.getRoot());
                treeA11.setRoot(treeA11.getCurrent());
                treeA22.setRoot(treeA22.getCurrent());

               NewTreeSetLeft(newtree);
               QTree.getLeft();treeA11=treecopy.CopyTree(QTree,treeA11);
               treeA11.setCurrent(treeA11.getRoot());
               treeA11.setNot();
               QTree.setCurrent(QTree.getRoot());
               inInfo(QTree,newtree);

               newtree.inData(pt.ParseTree(treeA11));
               treeA11.setCurrent(treeA11.getRoot());
               QTree.setCurrent(QTree.getRoot());
               newtree.inDataTree(treeA11); QTree.getLeft();
               newtree.inNodeTree(treeA11.getData());
               QTree.setCurrent(QTree.getRoot());
               //newtree.pretrav();
               QTree.getRight();treeA22=treecopy.CopyTree(QTree,treeA22);
               treeA22.setCurrent(treeA22.getRoot());
               QTree.setCurrent(QTree.getRoot());
               newtree.getParent();
               NewTreeSetRight(newtree);

               inInfo(QTree,newtree);
               newtree.inNodeTree(treeA22.getCurrent().getData());
               newtree.inData(pt.ParseTree(treeA22));
             }
           } //end Not == true
         newtree.pretrav();
         return newtree;

       } // end quantifier

//************************************************

        if( d.getOperator() != null) {

          //If the operator does not have a not
          if(d.getNot() == false)
              if("\u2227".equals(d.getOperator())) { // \u2227='and' operator
                 if( newtree.getCenter() == false) newtree.setCenter();
                    treeA=d;
                    treeA1.setRoot(treeA1.getCurrent());
                    treeA2.setRoot(treeA2.getCurrent());
                    treeA.setRoot(treeA.getCurrent());
                    treeAA.setRoot(treeAA.getCurrent());
                    treeAA=treecopy.CopyTree(treeA,treeAA);
                    treeA.setCurrent(treeA.getRoot());
                    treeAA.setCurrent(treeAA.getRoot());
                 NewTreeSetRight(newtree);
                 treeA.getLeft();

                 lleft = pt.ParseTree(treeA);
                 newtree.inLeftTree(treeA.getCurrent().getData());
                 treeA.setCurrent(treeA.getRoot()); treeA.getRight();
                 newtree.inRightTree(treeA2.getCurrent().getData());

                 rright =pt.ParseTree(treeA);treeA.setCurrent(treeA.getRoot());
                 newtree.inDataTree(treeAA);
                 treeRAA=treecopy.CopyTree(treeAA,treeRAA);
                 treeRAA.setCurrent(treeRAA.getRoot());
                 treeRAA.pretrav();
                 treeRAA.setCurrent(treeRAA.getRoot());
                 newtree.inRightDataTree(treeRAA);
```

```
  treeAA.setCurrent(treeAA.getRoot());
  treeLAA=treecopy.CopyTree(treeAA,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
  treeLAA.pretrav();
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inLeftDataTree(treeLAA);treeAA.setCurrent(treeAA.getRoot());
  newtree.inOperator("\u2194"); //\u2194= 'if and only if' operator
  newtree.inLeftData(lleft);System.out.println("41 lleft, "+lleft);
  newtree.inRightData(rright);System.out.println("42 rright,
  "+rright);
}// end \u2227 logical and

// Logical or operator
 else if("\u2228".equals(d.getOperator())) {//\u2228='or' operator
  if( newtree.getCenter() == true) newtree.setCenter();
     treeA=d; treeB=d;
     treeA.setRoot(treeA.getCurrent());
     treeA1.setRoot(treeA1.getCurrent());
     treeA2.setRoot(treeA2.getCurrent());

  NewTreeSetLeft(newtree);
  //newtree.pretrav();
  newtree.inOperator("\u2228");//\u2228='or' operator
  System.out.println("and 22 ");System.out.println(" ");
  treeA.getLeft();
  treeAA=treecopy.CopyTree(treeA,treeAA);treeAA.pretrav();
  treeA.setCurrent(treeA.getRoot());d.setCurrent(home);
  treeAA.setCurrent(treeAA.getRoot());

  newtree.inData(pt.ParseTree(treeAA));
  newtree.inDataTree(treeAA);treeAA.setCurrent(treeAA.getRoot());
  newtree.inNodeTree(treeAA.getCurrent().getData());
  inInfo(d,newtree);d.setCurrent(home); treeA.getRight();
  treeA.setCurrent(treeA.getRoot()); treeA.getRight();
  treeBB=treecopy.CopyTree(treeA,treeBB);treeBB.pretrav();
  newtree.getParent();

  NewTreeSetRight(newtree);
  inInfo(d,newtree);
  newtree.inData(pt.ParseTree(treeA));
  treeA2.setRoot(treeA2.getCurrent());
  newtree.inDataTree(treeBB);
  newtree.inNodeTree(treeBB.getData());
}// end \u2228 logical or

 // Logical if then operator
 else if("\u2192".equals(d.getOperator())) {//\u2192='if then
          operator'
  if( newtree.getCenter() == true) newtree.setCenter();
  treeA=d; treeB=d;
  treeA1.setRoot(treeA1.getCurrent());
  treeA2.setRoot(treeA2.getCurrent());
  treeA.setRoot(treeA.getCurrent());
  treeB.setRoot(treeB.getCurrent());

  NewTreeSetLeft(newtree);
  newtree.inOperator("\u2227"); //\u2227='or' operator
  treeA.getLeft();
  treeAA=treecopy.CopyTree(treeA,treeAA);
  treeA.setCurrent(treeA.getRoot()); treeA.getLeft();
  treeAA.setCurrent(treeAA.getRoot());
  newtree.setNot();//\u00AC= negation

  newtree.inData("\u00AC"+"("+pt.ParseTree(treeA)+")");
  treeAA.setNot();
  newtree.inDataTree(treeAA);
  newtree.inRightDataTree(treeAA);
```

```
  treeAA.setCurrent(treeAA.getRoot());
  treeAA.getCurrent().setNot();
  treeAA.setCurrent(treeAA.getRoot());

  newtree.inLeftDataTree(treeAA);
  newtree.inNodeTree(treeAA.getCurrent().getData());
  newtree.pretrav();
  treeA.setCurrent(treeA.getRoot()); treeA.getRight();
  newtree.getParent();
  treeBB=treecopy.CopyTree(treeA,treeBB);
  treeA.setCurrent(treeA.getRoot()); treeA.getRight();
  treeBB.setCurrent(treeBB.getRoot());
  newTreeSetRight(newtree);
  //inInfo(d,newtree);
  newtree.inData(pt.ParseTree(treeA));
  newtree.inDataTree(treeBB);
  newtree.inRightDataTree(treeBB);
  newtree.inLeftDataTree(treeBB);
  newtree.inNodeTree(treeBB.getCurrent().getData());
   newtree.pretrav();
}// end \u2192 logical and

//Logical if and only if operator
 else {
  if( newtree.getCenter() == true) newtree.setCenter();
  System.out.println("2 ");
  treeA=d;
  treeA1.setRoot(treeA1.getCurrent());
  treeA2.setRoot(treeA2.getCurrent());
  treeA.setRoot(treeA.getCurrent());
  treeAA.setRoot(treeAA.getCurrent());
  treeB.setRoot(treeB.getCurrent());
  treeAA=treecopy.CopyTree(treeA,treeAA);
  treeA.setCurrent(treeA.getRoot());
  treeBB=treecopy.CopyTree(treeA,treeBB);
  treeA.setCurrent(treeA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());
  treeBB.setCurrent(treeBB.getRoot());

  NewTreeSetLeft(newtree);
  treeA.getLeft();
  lleft = pt.ParseTree(treeA);
  newtree.inLeftTree(treeA.getCurrent().getData());
  treeA.setCurrent(treeA.getRoot()); treeA.getRight();
  newtree.inRightTree(treeA2.getCurrent().getData());
  rright =pt.ParseTree(treeA);treeA.setCurrent(treeA.getRoot());
  newtree.inDataTree(treeAA);
  treeRAA=treecopy.CopyTree(treeAA,treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.pretrav();
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.inRightDataTree(treeRAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeLAA=treecopy.CopyTree(treeAA,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
  treeLAA.pretrav();
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inLeftDataTree(treeLAA);treeAA.setCurrent(treeAA.getRoot());
  newtree.inOperator("\u2194");//\u2194='if and only if' operator
  newtree.inLeftData(lleft);
  newtree.inRightData(rright);
  newtree.getParent();

  NewTreeSetRight(newtree);
  newtree.setLNot();
  treeAA.setCurrent(treeAA.getRoot());
  treeAA.getLeft();treeAA.setNot();
```

```
                    newtree.inLeftData("\u00AC"+"("+lleft+")");//\u00AC=negation
                    newtree.inLeftTree(treeAA.getCurrent().getData());
                    newtree.setRNot();
                    treeAA.setCurrent(treeAA.getRoot()); //\u00AC=negation
                    newtree.inRightData("\u00AC"+"("+rright+")");
                    newtree.inRightTree(treeAA.getCurrent().getData());
                    newtree.inDataTree(treeBB);
                    treeRBB=treecopy.CopyTree(treeBB,treeRBB);
                    treeRBB.setCurrent(treeRBB.getRoot());
                    treeRBB.getRight();
                    treeRBB.setNot();treeRBB.setCurrent(treeRBB.getRoot());
                    newtree.inRightDataTree(treeRBB);
                    treeBB.setCurrent(treeBB.getRoot());
                    treeLBB=treecopy.CopyTree(treeBB,treeLBB);
                    treeLBB.setCurrent(treeLBB.getRoot());
                    treeLBB.getLeft(); treeLBB.setNot();
                    treeLBB.setCurrent(treeLBB.getRoot());
                    newtree.inLeftDataTree(treeLBB);treeBB.setCurrent(treeBB.getRoot());

                   }// end \u2194 logical if and only if

//*************************************************************
                  }// Not == false
                  else  { // Not == true
                  // Logical and operator
                   if("\u2227".equals(d.getOperator())) { //\u2227='and' operator
                     if( newtree.getCenter() == true) newtree.setCenter();
                      treeA=d; treeB=d;
                      treeA.setRoot(treeA.getCurrent());
                      treeB.setRoot(treeB.getCurrent());
                      treeA1.setRoot(treeA1.getCurrent());
                      treeA2.setRoot(treeA2.getCurrent());
                      treeA.getLeft();
                      treeAA=treecopy.CopyTree(treeA,treeAA);treeAA.pretrav();
                      lleft=pt.ParseTree(treeAA);
                      treeA.setCurrent(treeA.getRoot()); treeAA.setNot();
                      treeAA.setCurrent(treeAA.getRoot());
                      treeA.getRight();
                      rright=pt.ParseTree(treeA);  treeA.setCurrent(treeA.getRoot());

                      NewTreeSetLeft(newtree);
                      inInfo(d,newtree);//\u00AC=negation
                      newtree.inData("\u00AC"+"("+lleft+")");
                      treeAA.setCurrent(treeAA.getRoot());
                      newtree.inDataTree(treeAA);
                      newtree.inNodeTree(treeAA.getCurrent().getData());
                      newtree.getParent();
                      NewTreeSetRight(newtree);
                      inInfo(d,newtree);
                      treeB.getRight();
                      treeBB=treecopy.CopyTree(treeB,treeBB);treeBB.pretrav();
                      treeBB.setCurrent(treeBB.getRoot()); treeBB.setNot();
                      newtree.inData("\u00AC"+"("+rright+")");//\u00AC=negation
                      newtree.inDataTree(treeBB);
                      newtree.inNodeTree(treeBB.getCurrent().getData());

                   }// end \u2227 logical and

                   // Logical or operator
                    else if("\u2228".equals(d.getOperator())) {

                      if( newtree.getCenter() == false) newtree.setCenter();
                      treeA=d;
                      treeA1.setRoot(treeA1.getCurrent());
                      treeA2.setRoot(treeA2.getCurrent());
                      treeA.setRoot(treeA.getCurrent());
                      treeAA.setRoot(treeAA.getCurrent());
```

```
  treeB.setRoot(treeB.getCurrent());
  treeAA=treecopy.CopyTree(treeA,treeAA);
  treeA.setCurrent(treeA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());
  NewTreeSetRight(newtree);
  treeA.getLeft();
  newtree.inLeftTree(treeA.getCurrent().getData());
  treeAA.setCurrent(treeAA.getRoot()); treeAA.getRight();
  newtree.inRightTree(treeAA.getCurrent().getData());
  newtree.inDataTree(treeAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeRAA=treecopy.CopyTree(treeAA,treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.getRight(); treeRAA.getCurrent().setNot();
  treeRAA.setCurrent(treeRAA.getRoot());
  rright =pt.ParseTree(treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());treeRAA.pretrav();
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.inRightDataTree(treeRAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());
  treeLAA=treecopy.CopyTree(treeAA,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
  treeLAA.getLeft(); treeLAA.getCurrent().setNot();
  treeLAA.setCurrent(treeLAA.getRoot());
  lleft = pt.ParseTree(treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());treeLAA.pretrav();
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inLeftDataTree(treeLAA);treeAA.setCurrent(treeAA.getRoot());
  newtree.inOperator("\u2194");//\u2194='if and only if' operator
  treeLAA.getLeft();
  newtree.inLeftData(pt.ParseTree(treeLAA));
  treeLAA.setCurrent(treeLAA.getRoot()); treeRAA.getRight();
  newtree.inRightData(pt.ParseTree(treeRAA));
  treeRAA.setCurrent(treeRAA.getRoot());

}// end \u2228 logical Or

 // Logical if then operator
else if("\u2192".equals(d.getOperator())) {//\u2192='if then'
        operator
  if( newtree.getCenter() == false) newtree.setCenter();
  treeA=d;
  treeA1.setRoot(treeA1.getCurrent());
  treeA2.setRoot(treeA2.getCurrent());
  treeA.setRoot(treeA.getCurrent());
  treeAA.setRoot(treeAA.getCurrent());
  treeB.setRoot(treeB.getCurrent());
  treeAA=treecopy.CopyTree(treeA,treeAA);
  treeA.setCurrent(treeA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());
  NewTreeSetRight(newtree);
  treeA.getLeft();
  newtree.inLeftTree(treeA.getCurrent().getData());
  treeAA.setCurrent(treeAA.getRoot()); treeAA.getRight();
  newtree.inRightTree(treeAA.getCurrent().getData());
  newtree.inDataTree(treeAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeRAA=treecopy.CopyTree(treeAA,treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.getRight(); treeRAA.getCurrent().setNot();
  treeRAA.setCurrent(treeRAA.getRoot());
  rright =pt.ParseTree(treeRAA);treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.inRightDataTree(treeRAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());
```

```
  treeLAA=treecopy.CopyTree(treeAA,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
  treeLAA.getLeft();
  treeLAA.setCurrent(treeLAA.getRoot());
  lleft = pt.ParseTree(treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());treeLAA.pretrav();
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inLeftDataTree(treeLAA);treeAA.setCurrent(treeAA.getRoot());
  newtree.inOperator("\u2194");//\u2194='if an only if' operator
  treeLAA.getLeft();
  newtree.inLeftData(pt.ParseTree(treeLAA));
  treeLAA.setCurrent(treeLAA.getRoot()); treeRAA.getRight();
  newtree.inRightData(pt.ParseTree(treeRAA));
  treeRAA.setCurrent(treeRAA.getRoot());

}// end \u2192 logical if then

//Logical if and only if operator
 else {
                                                {
  if( newtree.getCenter() == true) newtree.setCenter();
  treeA=d;
  treeA1.setRoot(treeA1.getCurrent());
  treeA2.setRoot(treeA2.getCurrent());
  treeA.setRoot(treeA.getCurrent());
  treeAA.setRoot(treeAA.getCurrent());
  treeB.setRoot(treeB.getCurrent());
  treeAA=treecopy.CopyTree(treeA,treeAA);
  treeA.setCurrent(treeA.getRoot());
  treeBB=treecopy.CopyTree(treeA,treeBB);
  treeA.setCurrent(treeA.getRoot());
  treeAA.setCurrent(treeAA.getRoot());

  treeBB.setCurrent(treeBB.getRoot());
  NewTreeSetLeft(newtree);
  treeA.getLeft();
  lleft = pt.ParseTree(treeA);
  newtree.inLeftTree(treeA.getCurrent().getData());
  treeA.setCurrent(treeA.getRoot()); treeA.getRight();
  newtree.inRightTree(treeA2.getCurrent().getData());

  rright =pt.ParseTree(treeA);treeA.setCurrent(treeA.getRoot());
  newtree.inDataTree(treeAA);
  treeRAA=treecopy.CopyTree(treeAA,treeRAA);
  treeRAA.setCurrent(treeRAA.getRoot());
  treeRAA.getRight(); treeRAA.setNot();treeRAA.pretrav();
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.inRightDataTree(treeRAA);
  treeAA.setCurrent(treeAA.getRoot());
  treeLAA=treecopy.CopyTree(treeAA,treeLAA);
  treeLAA.setCurrent(treeLAA.getRoot());
  treeLAA.pretrav();
  treeLAA.setCurrent(treeLAA.getRoot());

  treeLAA.getLeft();
  newtree.inLeftData(pt.ParseTree(treeLAA));
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inLeftDataTree(treeLAA);treeAA.setCurrent(treeAA.getRoot());
  treeLAA.setCurrent(treeLAA.getRoot());
  newtree.inOperator("\u2194");//\u2194='if and only if' operato
  treeRAA.getRight();
  newtree.inRightData(pt.ParseTree(treeRAA));
  treeRAA.setCurrent(treeRAA.getRoot());
  newtree.getParent();
  NewTreeSetRight(newtree);
  treeAA.setCurrent(treeAA.getRoot()); treeAA.getLeft();
  newtree.inLeftTree(treeAA.getCurrent().getData());
```

```
                    treeAA.setCurrent(treeAA.getRoot());
                    newtree.inRightTree(treeAA.getCurrent().getData());
                    newtree.inDataTree(treeBB);
                    treeRBB=treecopy.CopyTree(treeBB,treeRBB);

                    treeRBB.getRight();
                    newtree.inRightData(pt.ParseTree(treeRBB));
                    treeRBB.setCurrent(treeRBB.getRoot());
                    newtree.inRightDataTree(treeRBB);
                    treeBB.setCurrent(treeBB.getRoot());
                    treeLBB=treecopy.CopyTree(treeBB,treeLBB);
                    treeLBB.setCurrent(treeLBB.getRoot());
                    treeLBB.getLeft(); treeLBB.setNot();
                    newtree.inLeftData(pt.ParseTree(treeLBB));
                    treeLBB.setCurrent(treeLBB.getRoot());
                    newtree.inLeftDataTree(treeLBB);
                    treeBB.setCurrent(treeBB.getRoot());

                  }// end \u2194 if and only if

                }
              newtree.pretrav();
              return newtree;
            }// end Operator

          newtree.pretrav();
          return newtree ;

        }// end Stree

         return newtree;
        }//try
    }
/*Since this program is version one, there's only going to be five branches
for the time being. This program can allow more branches later on.*/
// creates a new right node on the sematic tableaux tree and assigns a branch number.
public BinaryTree NewTreeSetRight(BinaryTree b){

       ChildNode tempBranch2 = new ChildNode();
       if(b.getCurrent().getData() != null || b.getCurrent().getRightData() !=null) {
                        tempBranch2=b.getCurrent();
                        if(b.getCurrent()==b.getBranch1()) {
                          tempBranch2= b.getCurrent();
                          b.insertRightGo();
                          b.setBranch1(b.getCurrent());
                          System.out.println(" "+"starting +A  sTree2");
                        }
                        else if(b.getCurrent()==b.getBranch2()) {
                          tempBranch2= b.getCurrent();
                          b.insertRightGo();
                          b.setBranch2(b.getCurrent());
                          System.out.println(" "+"starting +A  sTree2");
                        }
                        else if(b.getCurrent()==b.getBranch3()) {
                          tempBranch2= b.getCurrent();
                          b.insertRightGo();
                          b.setBranch3(b.getCurrent());
                          System.out.println(" "+"starting +A  sTree2");
                        }
                        else if(b.getCurrent()==b.getBranch4()) {
                          tempBranch2= b.getCurrent();
                          b.insertRightGo();
                          b.setBranch4(b.getCurrent());
                          System.out.println(" "+"starting +A  sTree2");
                        }
                        else {
                          tempBranch2= b.getCurrent();
```

```
                    b.insertRightGo();
                    b.setBranch5(b.getCurrent());
                    System.out.println(" "+"starting +A  sTree2");
                   }
                 }
               return b;
}

// creates a new left node on the sematic tableaux tree and assigns a branch number.
public BinaryTree NewTreeSetLeft(BinaryTree b){
        ChildNode tempBranch1 = new ChildNode();
        if(b.getCurrent()==b.getBranch1()) {
                    tempBranch1= b.getCurrent();
                    b.insertLeftGo();
                    b.setBranch1(b.getCurrent());
                    System.out.println(" "+"starting +A1l  sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch2()) {
                    tempBranch1=b.getCurrent();
                    b.insertLeftGo();
                    b.setBranch2(b.getCurrent());
                    System.out.println(" "+"starting +A2l  sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch3()) {
                    tempBranch1= b.getCurrent();
                    b.insertLeftGo();
                    b.setBranch3(b.getCurrent());
                    System.out.println(" "+"starting +A3l  sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch4()) {
                    tempBranch1=b.getCurrent();
                    b.insertLeftGo();
                    b.setBranch4(b.getCurrent());
                    System.out.println(" "+"starting +A4l  sTree2");
                   }
                   else {
                    tempBranch1= b.getCurrent();
                    b.insertLeftGo();
                    b.setBranch5(b.getCurrent());
                    System.out.println(" "+"starting +A5l  sTree2");
                   }
                  return b;
}
 //gets left node of a binary tree and assigns a branch number.
  public BinaryTree BranchsetLeft( BinaryTree b){

                 if(b.getCurrent()==b.getBranch1()) {
                    b.getLeft();
                    b.setBranch1(b.getCurrent());
                    System.out.println(" "+"starting +A 1 sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch2()) {
                    b.getLeft();
                    b.setBranch2(b.getCurrent());
                    System.out.println(" "+"starting +A2  sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch3()) {
                    b.getLeft();
                    b.setBranch3(b.getCurrent());
                    System.out.println(" "+"starting +A 3 sTree2");
                   }
                   else if(b.getCurrent()==b.getBranch4()) {
                    b.getLeft();
                    b.setBranch4(b.getCurrent());
                    System.out.println(" "+"starting +A4  sTree2");
                   }
                   else {
```

```
                    b.getLeft();
                    b.setBranch5(b.getCurrent());
                    System.out.println(" "+"starting +A 5 sTree2");
                   }

            return b;

   } // branchsetleft

//gets right node of a binary tree and assigns a branch number.
  public BinaryTree BranchsetRight( BinaryTree b) {
                   if(b.getRoot()==b.getBranch1()) {
                    b.getRight();
                    if(b.getData() == null)
                       b.getLeft();
                    b.setBranch1(b.getCurrent());
                    System.out.println(" "+"starting +A 1 sTree2");
                   }
                   else if(b.getRoot()==b.getBranch2()) {
                    b.getRight();
                    if(b.getData() == null)
                       b.getLeft();
                    b.setBranch2(b.getCurrent());
                    System.out.println(" "+"starting +A 2 sTree2");
                   }
                   else if(b.getRoot()==b.getBranch3()) {
                    b.getRight();
                    if(b.getData() == null)
                       b.getLeft();
                    b.setBranch3(b.getCurrent());
                    System.out.println(" "+"starting +A3  sTree2");
                   }
                   else if(b.getRoot()==b.getBranch4()) {
                    b.getRight();
                    if(b.getData() == null)
                       b.getLeft();
                    b.setBranch4(b.getCurrent());
                    System.out.println(" "+"starting +A 4 sTree2");
                   }
                   else {
                    b.getRight();
                    if(b.getData() == null)
                       b.getLeft();
                    b.setBranch5(b.getCurrent());
                    System.out.println(" "+"starting +A 5 sTree2");
                   }


                  return b;

  }

} // class STree

/*************************************************************
Name:   Interface.java
Description: GUI allows the user to input
Data: January 2005
*************************************************************/

package MS;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
```

```
public class Interface extends Applet{

        Panel p  ;
        Panel p1 =new Panel();
        Panel p2 = new Panel() ;
        TextField display =new TextField();
        TextField checkDisplay =new TextField();
        TextField OutDisplay = new TextField()
        CheckParse c = new CheckParse() ;
        String checkString = "" ;
        String getString = "" ;
        String A1 ="" ;
        String A2 ="" ;
        String con ="" ;
        String s ="";
        String s1="";
        Vector v = new Vector() ;
        ChildNode r = new ChildNode();
        Pa par = new Pa() ;
        BinaryTree t = new BinaryTree();
        BuildTree bt = new BuildTree();
        ParseTree pt = new ParseTree();
        ReplaceVariable rv = new ReplaceVariable();
        BinaryTree ts = new BinaryTree();
        BinaryTree ts1 = new BinaryTree();
        STree st = new STree();
        STree1 st1 = new STree1();
        Sending send = new Sending();

       public Interface()
        {
           GridBagLayout gridb1 = new GridBagLayout();
           GridBagLayout gridb2 = new GridBagLayout();
           GridBagConstraints constr = new GridBagConstraints();
           p1 = new Panel();
           p1.setLayout(gridb1) ;
           p2 = new Panel() ;
           p2.setLayout(gridb1) ;
           constr.fill = GridBagConstraints.BOTH;
           constr.weightx = 1.0;
           constr.anchor = GridBagConstraints.CENTER;
           display = new TextField(75);
           checkDisplay = new TextField(75);
           OutDisplay = new TextField(75);
           gridb1.setConstraints(display,constr);
           gridb1.setConstraints(checkDisplay,constr);
           gridb1.setConstraints(OutDisplay,constr);
           add(display);
           Button for_all = new Button("\u2200");
           gridb1.setConstraints(for_all,constr);
           add(for_all);
           Button for_all2 = new Button("For_All");
           gridb1.setConstraints(for_all2,constr);
           add(for_all2);
           Button there_exists = new Button("\u2203");
           gridb1.setConstraints(there_exists,constr);
           add(there_exists);
           Button there_exists2 = new Button("There_Exists");
           gridb1.setConstraints(there_exists2,constr);
           add(there_exists2);
           Button and = new Button("\u2227");
           gridb1.setConstraints(and,constr);
           add(and);
           Button and2 = new Button("And");
           gridb1.setConstraints(and2,constr);
           add(and2);
```

```
    Button or = new Button("\u2228");
    gridbl.setConstraints(or,constr);
    add(or);
    Button or2 = new Button("Or");
    gridbl.setConstraints(or2,constr);
    add(or2);
     Button negation = new Button("\u00AC");
    gridbl.setConstraints(negation,constr);
    add(negation);
    Button negation2 = new Button("Not");
    gridbl.setConstraints(negation2,constr);
    add(negation2);
    Button ifThen = new Button("\u2192");
    gridbl.setConstraints(ifThen,constr);
    add(ifThen);
    Button ifThen2 = new Button("If_Then");
    gridbl.setConstraints(ifThen2,constr);
    add(ifThen2);
    Button ifAndOnlyIf = new Button("\u2194");
    gridbl.setConstraints(ifAndOnlyIf,constr);
    add(ifAndOnlyIf);
    Button ifAndOnlyIf2 = new Button("If_And_Only_If");
    gridbl.setConstraints(ifAndOnlyIf2,constr);
    add(ifAndOnlyIf2);

    Button clear = new Button("Clear");
    gridbl.setConstraints(clear,constr);
    add(clear);
    add(checkDisplay);
    Button assump1 = new Button("Assumption1");
    gridbl.setConstraints(assump1,constr);
    add(assump1);
    Button assump2 = new Button("Assumption2");
    gridbl.setConstraints(assump2,constr);
    add(assump2);
    Button conclu = new Button("Conclusion");
    gridbl.setConstraints(conclu,constr);
    add(conclu);

    add(checkDisplay);

    Button correct = new Button("Correct");
    gridbl.setConstraints(assump2,constr);
    add(correct);
    Button incorrect = new Button("Incorrect");
    gridbl.setConstraints(conclu,constr);
    add(incorrect);

    add(OutDisplay);

    display.setText("");
    checkDisplay.setText("");
    checkString = null ;
    A1 = null ;
    A2 = null ;
    con = null ;
 }
 public boolean
action(Event event, Object ob)
{       checkString = null ;
                A1 = null ;
                A2 = null ;
                con = null ;
        if( event.target instanceof Button != "Clear".equals(ob) &&
            event.target instanceof Button != "Assumption1".equals(ob)&&
            event.target instanceof Button != "Assumption2".equals(ob)&&
            event.target instanceof Button != "Conclusion".equals(ob) &&
```

```
        event.target instanceof Button != "Correct".equals(ob)&&
        event.target instanceof Button != "Incorrect".equals(ob)&&
        event.target instanceof Button != "For_All".equals(ob) &&
        event.target instanceof Button != "There_Exists".equals(ob)&&
        event.target instanceof Button != "And".equals(ob)&&
        event.target instanceof Button != "Or".equals(ob) &&
        event.target instanceof Button != "Not".equals(ob)&&
        event.target instanceof Button != "If_Then".equals(ob)&&
        event.target instanceof Button != "If_And_Only_If".equals(ob) ){
       display.setText( display.getText() + ob );
       return true;
}
 else if ( "Clear".equals(ob) || "Incorrect".equals(ob)) {
            try{
            getString="";
            display.setText(null) ;
            clearA1();
            checkDisplay.setText(null);
            OutDisplay.setText(null);
            getString="";
            A1 = "";
            A2 ="" ;
            con ="" ;
            return true ;
            }
            catch(Exception e) {
            checkDisplay.setText("Unable to clear String");
            return false ;
            }
 }
 else if ( "For_All".equals(ob)) {
            display.setText( display.getText() + "\u2200" );
            return true ;
 }
 else if ( "There_Exists".equals(ob)) {
            display.setText( display.getText() + "\u2203" );
            return true ;
 }
 else if ( "And".equals(ob)) {
            display.setText( display.getText() + "\u2227" );
            return true ;
 }
 else if ( "Or".equals(ob)) {
            display.setText( display.getText() + "\u2228" );
            return true ;
 }
 else if ( "Not".equals(ob)) {
            display.setText( display.getText() + "\u00AC" );
            return true ;
 }
 else if ( "If_Then".equals(ob)) {
            display.setText( display.getText() + "\u2192" );
            return true ;
 }
 else if ( "If_And_Only_If".equals(ob)) {
            display.setText( display.getText() + "\u2194" );
            return true ;
 }
 else if ( "Assumption1".equals(ob)) {
            A1=null;
            try {
            A1=null;
            getString = display.getText() ;
            checkString=c.CheckParse(getString) ;
            checkDisplay.setText(checkString);
            A1 = display.getText();
            send.setA1(A1);
```

```
                    System.out.println(send.getA1());
                    return true ;
                    }
                    catch(Exception e) {
                    checkDisplay.setText("Unable to Check String");
                    OutDisplay.setText("String entered was incorrect. Try
                     again."+t.getOperator()+t.getData());
                    return false ;
                    }
                }
                else if ( "Assumption2".equals(ob)) {
                    try {
                    checkString=" ";
                    A2=" ";
                    checkString=c.CheckParse(display.getText()) ;
                    checkDisplay.setText(checkString);
                    OutDisplay.setText("If this is correct, press 'Correct'. If
                    needed re-load the page and try again.");
                    A2 = display.getText();
                    send.setA2(A2);
                    return true ;
                       }
                    catch(Exception e) {
                    checkDisplay.setText("Unable to Check String");
                    OutDisplay.setText("String entered was incorrect. Try again.");
                    return false ;
                    }
                }
                else if ( "Conclusion".equals(ob)) {
                    try {
                       checkString=c.CheckParse(display.getText()) ;
                       checkDisplay.setText(checkString);
                       con = display.getText();
                       OutDisplay.setText("Did you remember to negate this WFF ? If
                       correct, press 'correct'.");
                       send.setC1(con);
                       return true ;
                       }//try
                    catch(Exception e) {
                       checkDisplay.setText("Unable to Check String");
                       return false ;
                    }//catch
                }
               else return false;
        }
        public String getA1(String a) {
          return a ;
        }
        public String getA2(String a){
          return a ;
        }
        public String getCon(String a){
           return a;
        }
        public void clearA1(){
           A1="";
        }
}

/*************************************************************
Name:   Output.java
Description:   Output GUI to create Semantic Tableaux tree.
Data: January 2006
*************************************************************/


package MS;
```

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Output extends Applet{

  private String inputAA="";
  private String inputBB="";
  TextField display =new TextField();
  TextField inputA = new TextField();
  TextField inputB = new TextField();
  String inA ="";
  String inB ="";
  String inCatch ="" ;
  String enter ="";

  int mousex=0, mousey=0;
  int slength=0;
  int finalTx=0, finalTy=0;
  int workx = 0, worky=0;
  int b1x=0, b1y=0;
  int bl1x=0, bl1y =0;
  int br1x=0, br1y =0;
  int b2x=0, b2y=0;
  int bl2x=0, bl2y =0;
  int br2x=0, br2y =0;
  int b3x=0, b3y=0;
  int bl3x=0, bl3y =0;
  int br3x=0, br3y =0;
  int b4x=0, b4y=0;
  int bl4x=0, bl4y =0;
  int br4x=0, br4y =0;
  int b5x=0, b5y=0;
  int bl5x=0, bl5y =0;
  int br5x=0, br5y =0;
  int b6x=0, b6y=0;
  int bl6x=0, bl6y =0;
  int br6x=0, br6y =0;
  int b7x=0, b7y=0;
  int bl7x=0, bl7y =0;
  int br7x=0, br7y =0;
  int b8x=0, b8y=0;
  int bl8x=0, bl8y =0;
  int br8x=0, br8y =0;
  int b9x=0, b9y=0;
  int bl9x=0, bl9y =0;
  int br9x=0, br9y =0;
  int b10x=0, b10y=0;
  int bl10x=0, bl10y =0;
  int br10x=0, br10y =0;
  int b11x=0, b11y=0;
  int bl11x=0, bl11y =0;
  int br11x=0, br11y =0;

   String currentVar1a= "";
   String currentVar1b= "";
   String currentVar2a= "";
   String currentVar2b= "";
   String currentVar3a= "";
   String currentVar3b= "";
   String pred="";
   String pv1="";
   String pv2="";
```

```
String s;
Vector v1 = new Vector() ;
Vector v2 = new Vector();
Vector v3 = new Vector();
ChildNode root1 = new ChildNode();
ChildNode root2 = new ChildNode();
ChildNode root3 = new ChildNode();
ChildNode tempRoot1 = new ChildNode();
ChildNode tempCurrent = new ChildNode();
ChildNode finalRoot = new ChildNode();
ChildNode wdtRoot = new ChildNode();
ChildNode wtRoot = new ChildNode();
ChildNode hf = new ChildNode();
ChildNode finish = new ChildNode();
Pa parse1 = new Pa() ;
Pa parse2 = new Pa() ;
Pa parse3 = new Pa() ;
BinaryTree temp1 = new BinaryTree();
BinaryTree t1 = new BinaryTree();
BinaryTree t2 = new BinaryTree();
BinaryTree t3 = new BinaryTree();
BinaryTree finalT = new BinaryTree();
BinaryTree currentT = new BinaryTree();
BinaryTree workDataT = new BinaryTree();
BinaryTree workT = new BinaryTree();
BinaryTree drawT = new BinaryTree();
BinaryTree helpfinal=new BinaryTree();
BuildTree buildT1 = new BuildTree();
BuildTree buildT2 = new BuildTree();
BuildTree buildT3 = new BuildTree();
ParseTree parseT = new ParseTree();
CopyTree treecopy = new CopyTree();
SearchTree search = new SearchTree();
CaptureTree cap = new CaptureTree();
ReplaceVariable rv = new ReplaceVariable();
ReplaceCapture rc=new ReplaceCapture();
String searchS1="";
String searchS2="";
String captureS1="";
String captureS2="";
boolean start = true;
boolean startOne=false;
boolean startTwo=false;
boolean startThree=false;
boolean pickVar = false;
boolean pickOne = false;
boolean pickTwo = false;
boolean pickThree=false;
boolean pickEnter=false;
boolean anotherVar=false;
boolean pickedworkxy=false;
boolean bottomLeft=false;
boolean travelLeft=false;
boolean closedNot=false;
boolean Cloop=true;
boolean Cloop2=true;
boolean t1Closed=false;
boolean t2Closed=false;
boolean t3Closed=false;
boolean b6LCheckedOff=false;
boolean b6RCheckedOff=false;
boolean t1click=false;
boolean t2click=false;
boolean t3click=false;

STree st = new STree();
Sending getting = new Sending();
```

```
public void init() {

        GridBagLayout gridb1 = new GridBagLayout();
        GridBagLayout gridb2 = new GridBagLayout();
        GridBagConstraints constr = new GridBagConstraints();
        constr.fill = GridBagConstraints.BOTH;
        constr.weightx = 1.0;

        constr.fill = GridBagConstraints.BOTH;
        constr.weightx = 1.0;

        display = new TextField(135);
        gridb1.setConstraints(display,constr);
        add(display);
        display.setText(Sending.getA1());

        Button one = new Button("One");
        add(one);
        Button two = new Button("Two");
        add(two);
        Button three = new Button("Three");
        add(three);
        inputA = new TextField(5);
        gridb1.setConstraints(inputA,constr);
        add(inputA);
        inputB = new TextField(5);
        gridb1.setConstraints(inputB,constr);
        add(inputB);
        Button enter = new Button("Enter");
        add(enter);

        try {
        System.out.println(getting.getA1());

        start1=getting.getA1();
        sart2=getting.getA2();
        start3=getting.getC1();
        workDataT.setRoot(wdtRoot);
        workT.setRoot(wtRoot);
        helpfinal.setRoot(hf);
        t1.setRoot(root1);
        t2.setRoot(root2);
        t3.setRoot(root3);
        temp1.setRoot(tempRoot1);
        v1=parse1.Pa(start1);
        t1=buildT1.BuildTree(v1);
        t1.setCurrent(t1.getRoot());

        v2=parse2.Pa(start2);
        t2=buildT2.BuildTree(v2);
        t2.setCurrent(t2.getRoot());

        v3=parse3.Pa(start3);
        t3=buildT3.BuildTree(v3);
        t3.setCurrent(t3.getRoot());

        currentVar1a=t1.getRoot().getVariable1();
        currentVar2a=t2.getRoot().getVariable1();
        currentVar3a=t3.getRoot().getVariable1();
        finalT.setRoot(finalRoot);

        display.setText("where do you want to start from?");

        inputA.setText("");
        inputB.setText("");
```

```
        }
        catch( Exception e) {
        System.out.println("wrong");

       }

}


public boolean action(Event event, Object ob) {

         if(event.target instanceof Button =="One".equals(ob) ) {
           try{
                if(start==true){
                    if(t1.getData() == null){
                       if(t1.getCurrent().getLeft() !=null){
                           t1.getLeft();
                       }
                       else
                           t1.getRight();
                    }

                    if(t1.getQuantifier() != null) {
                       if(isCheckedOff(t1) == false) t1.getCurrent().setCheckedOff();

                       display.setText("Pick a variable or variables in the two input
                       boxes and then press enter");

                       start=false;
                       pickOne=true;
                       }
                      else {
                        if(isCheckedOff(t1) == true) {
                          t1.getCurrent().setCheckedOff();
                          t1Closed=true;
                          }
                            st.STree(finalT,t1,"*","*");
                            finalT=doSTree1();
                            finalT=doSTree2();
                            finalT=doSTree3();
                            finalT=doSTree4();
                            finalT=doSTree5();
                            repaint();
                            pickOne=false;
                            start=false;
                      }
                    repaint();
                }// if start=true
               else{
                  t1.setClicked();
                  if(t1.getData() == null){display.setText(t1.getData());
                       if(t1.getCurrent().getLeft() !=null){ //getting left
                           t1.getLeft();
                       }
                       else
                       t1.getRight();
                    }
                  display.setText("Click on the area of tree you wish to continue on
                  and press 'Enter'. Remember to put in the needed variables if
                  needed.");
                  pickOne=true;
                  repaint();
              }// start =false
           }
           catch(Exception e) {
```

```
    }

     return true;
  }
 else if("Two".equals(ob) ) {
     try{
        if(start==true){
             if(t2.getData() == null){
                if(t2.getCurrent().Left() !=null){
                   t2.getLeft()
                }
                else
                    t1.getRight();
             }

             if(isCheckedOff(t2) == false) t2.getCurrent().setCheckedOff();
             if(t2.getQuantifier() != null) {
                if(isCheckedOff(t2) == false) t2.getCurrent().setCheckedOff();
                    display.setText("Pick a variable or variables in the two
                    input boxes and then press enter");
                    start=false;
                    pickTwo=true;
                }
               else {
                 if(isCheckedOff(t2) == true) {
                   t2.getCurrent().setCheckedOff();
                   t2Closed=true;
                  }
                     st.STree(finalT,t2,"*","*");
                     finalT=doSTree1();
                     finalT=doSTree2();
                     finalT=doSTree3();
                     finalT=doSTree4();
                     finalT=doSTree5();
                     repaint();
                     display.setText("Click on the tree you wish to continue
                     and press 'Enter',or select another formula.
                     Remember to select variables if needed.");
                     pickTwo=false;
                     start=false;
               }
             repaint();
        }// if start=true
       else{ t2.setClicked();

        if(t2.getData() == null){
              if(t2.getCurrent().getLeft() !=null){
                 t2.getLeft();
               }
               else t2.getRight();
        }
        display.setText("Click on the area of tree you wish to continue on
        and press 'enter'. Remember to put in the
        needed variables if needed.");
        pickTwo=true;
        repaint();
      }// start =false
   }
   catch(Exception e) {

   }
    return true;
 }
 else if("Three".equals(ob) ) {
     try{
       if(start==true){
```

```
        if(t3.getData() == null){
           if(t3.getCurrent().getLeft() !=null){
              t3.getLeft();
           }
           else
              t3.getRight();
         }   if(t3.getCheckedOff() == false)
                t3.getCurrent().setCheckedOff();
        if(t3.getQuantifier() != null) {
           display.setText("Pick a variable or variables in the two input
           boxes and then press enter");
           start=false;
           pickThree=true;
           }
          else {
             if(isCheckedOff(t3) == true) {
              t3.getCurrent().setCheckedOff();
              t3Closed=true;
             }
             st.STree(finalT,t3,"*","*");

             finalT=doSTree1();
             finalT=doSTree2();
             finalT=doSTree3();
             finalT=doSTree4();
             finalT=doSTree5();

             repaint();

            display.setText(Click on the tree you wish to continue and
             press 'Enter',or select another formula. Remember to select
             variables if needed.");
             pickThree=false;
             start=false;

           }

          repaint();
       }// if start=true
       else{
          if(t3.getCheckedOff() == true) t3.getCurrent().setCheckedOff();
          t3.setClicked();

          if(t3.getData() == null){display.setText(t3.getData());
               if(t3.getCurrent().getLeft() !=null){
                  t3.getLeft();
               }
               else
                  t3.getRight();
            }
          display.setText("Click on the area of tree you wish to continue on
          and press the enter. Remember to put in the
          needed variables if needed.");
          pickThree=true;
          repaint();
      }// start =false
  }
  catch(Exception e) {

  }

   return true;
}
else if("Enter".equals(ob) ) {

 try {
```

```
if(pickOne ==true){
  inA=inputA.getText();
  inB=inputB.getText();
  this.inputAA = inA ;
  this.inputBB=inB;
  finalTx=mousex; finalTy=mousey;
  finalT=currentFinalT(finalTx, finalTy);
  if(t1.getData() == null)
     t1.getLeft();

  if(isCheckedOff(t1) == false)  t1.getCurrent().setCheckedOff();
  else t1.setClicked();

  if(t1.getQuantifier() != null) {

    if(Character.isLowerCase(inputAA.charAt(0))||
       Character.isLowerCase(inputBB.charAt(0))||){

       if(checkOne(inA,t1)==false){
      if(inB.length()==0){
           inB="BLANK";
         }
       else{
           searchS2=search.SearchTree(t1,inB);
           t1.setCurrent(t1.getRoot());
           if(t1.getData() == null) t1.getLeft();
           captureS2=cap.CaptureTree(t1);
           t1.setCurrent(t1.getRoot());
           if(t1.getData() == null) t1.getLeft();
           t1.getRight();
           t1=rc.ReplaceCapture(t1,searchS2,captureS2);
           t1.pretrav();
           t1.setCurrent(t1.getRoot());
           if(t1.getData() == null) t1.getLeft();
       }

     searchS1=search.SearchTree(t1,inA);

    if(searchS1.equals(inA) ){
        t1.setCurrent(t1.getRoot());
           if(t1.getData() == null)
               t1.getLeft();
           captureS1=cap.CaptureTree(t1);
           t1.setCurrent(t1.getRoot());
           if(t1.getData() == null)
              t1.getLeft();
           t1.getRight();
           t1=rc.ReplaceCapture(t1,searchS1,captureS1);
           t1.setCurrent(t1.getRoot());
           if(t1.getData() == null)
              t1.getLeft();
           if(t1.getQuantifier().equals("1")) {}
            else var.addElement(inA);
          st.STree(finalT,t1,inA,inB);
          pickOne=false;
          finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
          mousex=mousex-mousex;mousey=mousey-mouse;
          display.setText("'"+searchS1+"'"+" was replaced by
          "+"'"+captureS1+"'"+" because of capture was possible.
          Select or click a formula to continue.
          Remember to select variables if needed.");
          inputA.setText("");
          inputB.setText("");
          repaint();

       }
       else {
```

```
        t1.setCurrent(t1.getRoot());
        if(t1.getData() == null)
           t1.getLeft();
         if(t1.getQuantifier().equals("1") ){}
          else var.addElement(inA);
          if(t1.getQuantifier().equals("1")) {}
          else {
            if(inB.length()!=0)
               var.addElement(inB);
          }
         st.STree(finalT,t1,inA,inB);
         display.setText("pickone"+inB+inA);
         finalT=doSTree1();
         finalT=doSTree2();
         finalT=doSTree3();
         finalT=doSTree4();
         finalT=doSTree5();
         pickOne=false;
         repaint();
         finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
         mousex=mousex-mousex;mousey=mousey-mouse;
         inputA.setText("");
         inputB.setText("");
         repaint();
        }
     else {
        display.setText("Try again. Use a new
        variable.");pickOne=false;
        if(t1.getClicked()==false) t1.setClicked();
           t1Closed=false;
     }
    }//lowercase
   else{
        display.setText("Error. Wrong entry, try
         again.");pickOne=false;
         if(t1.getClicked()==false) t1.setClicked();
            t1Closed=false;

   }
  }
  else {
   st.STree(finalT,t1,"BLANK","BLANK");
     finalT=doSTree1();
     finalT=doSTree2();
     finalT=doSTree3();
     finalT=doSTree4();
     finalT=doSTree5();
     repaint();
  pickOne=false;
  repaint();finalT=closedT(finalT);
  finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
  mousex=mousex-mousex;mousey=mousey-mousey;
  display.setText("Click on the formula on the tree you wish to
  continue and press 'Enter',or click on the tree
  you wish to continue on and select another formula.
   Remember to select variables if needed.");
  inputA.setText("");
  inputB.setText("");
  repaint();

  }
  if(isCheckedOff(t1) == false) t1.setClicked();
}
else if(pickTwo ==true){
   inA=inputA.getText();
   inB=inputB.getText();
   this.inputAA =inA ;
```

```
this.inputBB=inB;
t2.setCurrent(t2.getRoot());
if(t2.getData() == null)  t2.getLeft();
finalTx=mousex; finalTy=mousey;
finalT=currentFinalT(finalTx, finalTy);
if(isCheckedOff(t2) == false)  t2.getCurrent().setCheckedOff();
else t2.setClicked();
if(t2.getQuantifier() != null) {

  if(Character.isLowerCase(inputAA.charAt(0))
      || Character.isLowerCase(input BB.charAt(0))){{

    if(checkOne(inA,t2)==false){
    if(inB.length()==0){
         inB="BLANK";
       }
     else{
         searchS2=search.SearchTree(t2,inB);
         t2.setCurrent(t2.getRoot());
         if(t2.getData() == null)  t2.getLeft();
         captureS2=cap.CaptureTree(t2);
         t2.setCurrent(t2.getRoot());
         if(t2.getData() == null) t2.getLeft();
         t2.getRight();
         t2=rc.ReplaceCapture(t2,searchS2,captureS2);
         t2.pretrav();
         t2.setCurrent(t2.getRoot());
         if(t2.getData() == null) t2.getLeft();
      }
    searchS1=search.SearchTree(t2,inA);
    if(searchS1.equals(inA) ){

       t2.setCurrent(t2.getRoot());
       if(t2.getData() == null) t2.getLeft();
       captureS1=cap.CaptureTree(t2);
       t2.setCurrent(t2.getRoot());
       if(t2.getData() == null) t2.getLeft();
       t2.getRight();
       t2=rc.ReplaceCapture(t2,searchS1,captureS1);
       t2.pretrav();
       t2.setCurrent(t2.getRoot());
       if(t2.getData() == null)  t2.getLeft();
       if(t2.getQuantifier().equals("1")) {}
         else var.addElement(inA);
       st.STree(finalT,t2,inA,inB);
       pickTwo=false;
       finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
       mousex=mousex-mousex;mousey=mousey-mousey;
       display.setText("'"+searchS1+"'"+" was replaced by
       "+"'"+captureS1+"'"+" because of capture was possible.
       Select or click a formula to continue.
       Remember to select variables if needed.");
       inputA.setText("");
       inputB.setText("");
       repaint();
      }
     else {
       t2.setCurrent(t2.getRoot());
       if(t2.getData() == null)  t2.getLeft();
       if(t2.getQuantifier().equals("1") ){}
         else var.addElement(inA);
         if(t2.getQuantifier().equals("1")) {}
         else {
           if(inB.length()!=0)
              var.addElement(inB);
         }
       st.STree(finalT,t2,inA,inB);
```

```
        finalT=doSTree1();
        finalT=doSTree2();
        finalT=doSTree3();
        finalT=doSTree4();
        finalT=doSTree5();
        pickTwo=false;;
        repaint();
        finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
        mousex=mousex-mousex;mousey=mousey-mousey;
        display.setText("Click on the formula on the tree you wish to
        continue and press 'Enter',or click on the tree you
        wish to continue on and select other formula.
        Remember to select variables if needed.");
        inputA.setText("");
        inputB.setText("");
        repaint();
        }
   else{
         display.setText("Try again. Pick another variable.");
         pickTwo=false;
        if(t2.getClicked()==false) t2.setClicked();
           t2Closed=false;
    }
   }//uppercase
  else{
         display.setText("Error. Wrong entry.");
         pickTwo=false;
        if(t2.getClicked()==false) t2.setClicked();
           t2Closed=false;
  }
 }
 else {
     st.STree(finalT,t2,"BLANK","BLANK");
     finalT=doSTree1();
     finalT=doSTree2();
     finalT=doSTree3();
     finalT=doSTree4();
     finalT=doSTree5();
     pickTwo=false;
     repaint();
     finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
     mousex=mousex-mousex;mousey=mousey-mousey
     inputA.setText("");
     inputB.setText("");
     repaint();
 }

}
else if(pickThree ==true){
 inA=inputA.getText();
 inB=inputB.getText();
 this.inputAA=inA;
 this.inputBB=inB;
 if(t3.getData() == null) t3.getLeft();
 finalTx=mousex; finalTy=mousey;
 finalT=currentFinalT(finalTx, finalTy);
 if(isCheckedOff(t3) == false) t3.getCurrent().setCheckedOff();
 else t3.setClicked();
 if(t3.getCurrent().getQuantifier() != null) {
  if(Character.isLowerCase(inputAA.charAt(0))
     || Character.isLowerCase(inputBB.charAt(0))){
    if(checkOne(inA,t3)==false){
     if(inB.length()==0){
         inB="BLANK";
     }
     else{
          searchS2=search.SearchTree(t3,inB);
```

```
        t3.setCurrent(t3.getRoot());
        if(t3.getData() == null)  t3.getLeft();
        captureS2=cap.CaptureTree(t3);
        t3.setCurrent(t3.getRoot());
        if(t3.getData() == null)  t3.getLeft();
        t3.getRight();
        t3=rc.ReplaceCapture(t3,searchS2,captureS2);
        t3.pretrav();
        t3.setCurrent(t3.getRoot());
        if(t3.getData() == null)   t3.getLeft();
    }
  t3.setCurrent(t3.getRoot());
  if(t3.getData()==null) t3.getLeft();
  t3.getRight();
  searchS1=search.SearchTree(t3,inA);
  if(searchS1.equals(inA)){
       t3.setCurrent(t3.getRoot());
       if(t3.getData() == null)  t3.getLeft();
       captureS1=cap.CaptureTree(t3);
       t3.setCurrent(t3.getRoot());
       if(t3.getData() == null) t3.getLeft();
       t3.getRight();
       t3=rc.ReplaceCapture(t3,searchS1,captureS1);
       t3.pretrav();
       t3.setCurrent(t3.getRoot());
       if(t3.getData() == null) t3.getLeft();
       if(t3.getQuantifier().equals("1")) {}
        else var.addElement(inA);
       st.STree(finalT,t3,inA,inB);
       pickThree=false;
       finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
       mousex=mousex-mousex;mousey=mousey-mousey;
       display.setText("'"+searchS1+"'"+" was replaced by
       "+"'"+captureS1+"'"+" because of capture was possible.
       Select or click a formula to continue.
       Remember to select variables if needed.");
       inputA.setText("");
       inputB.setText("");
       repaint();
    }
    else {
      t3.setCurrent(t3.getRoot());
      if(t3.getData() == null)
         t3.getLeft();
      if(t3.getQuantifier().equals("1") ){}
        else var.addElement(inA);
      if(t3.getQuantifier().equals("1")) {}
      else {
          if(inB.length()!=0)
             var.addElement(inB);
        }
      st.STree(finalT,t3,inA,inB);
      finalT=doSTree1();
      finalT=doSTree2();
      finalT=doSTree3();
      finalT=doSTree4();
      finalT=doSTree5();
      display.setText(finalT.getLeftData());
      pickThree=false;
      repaint();
      finalT=closedT(finalT);
      finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
      mousex=mousex-mousex;mousey=mousey-mousey;
      display.setText(finalT.getLeftData()+"Click on the formula on
      the tree you wish to continue and press 'Enter',or
      click on the tree you wish to continue on and select another
      formula. Remember to
```

```
        select variables if needed.");
        inputA.setText("");
        inputB.setText("");
        repaint();
       }
    else{
         display.setText("Try again. Pick another variable.");
         pickThree=false;
        if(t3.getClicked()==false) t3.setClicked();
           t3Closed=false;
     }
    }//uppercase
    else{
        display.setText("Error. Wrong entry, try
        again.");pickThree=false;
         if(t3.getClicked()==false) t3.setClicked();
           t3Closed=false;
   }
  }
  else {
   st.STree(finalT,t3,"BLANK","BLANK");
   finalT=doSTree1();
   finalT=doSTree2();
   finalT=doSTree3();
   finalT=doSTree4();
   finalT=doSTree5();
   pickThree=false;
   repaint();
   finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
   mousex=mousex-mousex;mousey=mousey-mousey;
   display.setText("Click on the formula on the tree you wish to
   continue and press 'Enter',or click on the tree you wish to
   continue on and select another formula. Remember
   to select variables if needed.");
   inputA.setText("");
   inputB.setText("");
   repaint();
  }
}
else if(pickedworkxy == false){
   workx=mousex; worky=mousey;
   mousex=mousex-mousex; mousey=mousey-mousey;
   workDataT=getBranch(workx,worky); //get datatree()
   finalTx=workx;finalTy=worky;
   finalT=(currentFinalT(workx,worky));
   if(finalT.getRightData() != null || finalT.getLeftData()){
      if (finalT.getBranch6() != finalT.getCurrent())
           finalT.setBranch6(finalT.getCurrent());
      else if (finalT.getBranch7() != finalT.getCurrent())
           finalT.setBranch7(finalT.getCurrent());
      else {//do nothing}
    }
   if(finalT.getData() != null){
      if(isCheckedOff(finalT)==false){
         if(finalT.getBranch8() != finalt.getRoot()
            finalT.setBranch8(finalT.getCurrent());
         else if(finalT.getBranch9() != finalt.getRoot()
                  finalT.setBranch9(finalT.getCurrent());
         else if(finalT.getBranch10() != finalt.getRoot()
                  finalT.setBranch10(finalT.getCurrent());
         else if(finalT.getBranch11() != finalt.getRoot()
                  finalT.setBranch11finalT.getCurrent());
         else{//do nothing}
         }
    }
   if(getDirection(workx,worky).equals("left"))
      {finalT.setLClicked();}
```

```
    else if(getDirection(workx,worky).equals("right"))
        {finalT.setRClicked();}
    else {finalT.setClicked();}
    repaint();
    display.setText("Click on the area of tree you wish to continue on
    and press enter. Remember to replace
    variable if needed.");
    pickedworkxy = true;
}
else if (pickedworkxy == true){
    inA=inputA.getText();
    inB=inputB.getText();
    this.inputAA=inA;
    this.inputBB=inB;
    finalTx=mousex; finalTy=mousey;
    mousex=mousex-mousex;mousey=mousey-mousey
    finalT=currentFinalT(finalTx, finalTy);
    String enter = getBranchxy(workx,worky);
    workDataT=getBranch(workx,worky); //get datatree()

    if(getDirection(workx,worky).equals("right")){
        workDataT.getRight();
        if(workDataT.getPredicate() != null ||
           workDataT.getQuantifier() == "0")
            workDataT.getParent();
        if(isCheckedOff(workDataT)==true) finalT.setRCheckedOff();
        else finalT.setRClicked();
        if(workDataT.getQuantifier() != null){
          if(Character.isLowerCase(inputAA.charAt(0))||
             Character.isLowerCase(inputBB.charAt(0))||
             ){
            if(inB.length()==0){
                inB="BLANK";
            }
            else{
                 searchS2=search.SearchTree(workDataT,inB);
                 workDataT.setCurrent(workDataT.getRoot());
                 if(workDataT.getData() == null)
                    workDataT.getLeft();
                 captureS2=cap.CaptureTree(workDataT);
                 workDataT.setCurrent(workDataT.getRoot());
                 if(workDataT.getData() == null)
                    workDataT.getLeft();
                 workDataT.getRight();
                 workDataT=rc.ReplaceCapture(workDataT,searchS2,
                 captureS2);
                 workDataT.pretrav();
                 workDataT.setCurrent(workDataT.getRoot());
                 if(workDataT.getData() == null) workDataT.getLeft();
                 workDataT.getRight();
               }//searchS1="1";

          searchS1=search.SearchTree(workDataT,inA);
          if(searchS1.equals(inA) ){
              workDataT.setCurrent(workDataT.getRoot());
              if(workDataT.getData() == null) workDataT.getLeft();
                  workDataT.getRight();
              captureS1=cap.CaptureTree(workDataT);
              workDataT.setCurrent(workDataT.getRoot());
              if(workDataT.getData() == null) workDataT.getLeft();
                  workDataT.getRight();
              workDataT.getRight();
              workDataT=rc.ReplaceCapture(workDataT,searchS1,
              captureS1);
              workDataT.pretrav();
              workDataT.setCurrent(workDataT.getRoot());
              if(workDataT.getData() == null)    workDataT.getLeft();
```

```
            workDataT.getRight();
          }
        if(CheckQ(IDBranch,finalT) == true) {
            finalT=currentFinalT(finalTx, finalTy);
            if(workDataT.getQuantifier().equals("1")) {}
            else {
               var.addElement(inA);
               if(inB.length()!=0)
                 var.addElement(inB);
             }

          st.STree(finalT,workDataT,inA,inB);
          finalT=doSTree1();
          finalT=doSTree2();
          finalT=doSTree3();
          finalT=doSTree4();
          finalT=doSTree5();
          finalT=currentFinalT(workx,worky);
        }

       else{
                display.setText("Incorrect place, try again");
                finalT=currentFinalT(workx,worky);
                if(finalT.getRClicked()==true) finalT.setRClicked();
                pickedworkxy=false;
                finalT=currentFinalT(finalTx, finalTy);repaint();

            }
        }//lowercase

       else{
           display.setText("Try again. Pick another variable.");
           if(finalT.getRClicked()==false) finalT.setRClicked();
             finalT.setRClosed();
           pickedworkxy=false;

       }

     }
   else{
        st.STree(finalT,workDataT,"BLANK","BLANK");
        finalT=currentFinalT(workx,worky);
        workDataT.getParent();finalT.inDataTree(workDataT);
        finalT=doSTree1();
        finalT=doSTree2();
        finalT=doSTree3();
        finalT=doSTree4();
        finalT=doSTree5();
        inputA.setText("");
        inputB.setText("");
        workx=workx-workx;worky=worky-worky;
        finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
        mousex=mousex-mousex;mousey=mousey-mousey;
        repaint(); pickedworkxy=false;
   }
   else if(getDirection(workx,worky).equals("left")){
        workDataT.getLeft();
        if(isCheckedOff(workDataT)==true) finalT.setLCheckedOff();
        else finalT.setLClicked();
        if(workDataT.getPredicate() != null){
            workDataT.setCurrent(workDataT.getRoot());
        }
        if(workDataT.getQuantifier() != null){
         if(Character.isLowerCase(inputAA.charAt(0))||
         Character.isLowerCase(inputBB.charAt(0))||
         ){
            if(inB.length()==0){
```

```
            inB="BLANK";
        }
         else{
            searchS2=search.SearchTree(workDataT,inB);
            workDataT.setCurrent(workDataT.getRoot());
            if(workDataT.getData() == null)
               workDataT.getLeft();
            workDataT.getLeft();
            captureS2=cap.CaptureTree(workDataT);
            workDataT.setCurrent(workDataT.getRoot());
            if(workDataT.getData() == null)
               workDataT.getLeft();
            workDataT.getLeft();
            workDataT=
            rc.ReplaceCapture(workDataT,searchS2,captureS2);
            workDataT.pretrav();
            workDataT.setCurrent(workDataT.getRoot());
            if(workDataT.getData() == null)
               workDataT.getLeft();
            workDataT.getLeft();
        }
     searchS1=search.SearchTree(workDataT,inA);
     if(searchS1.equals(inA) ){
         workDataT.setCurrent(workDataT.getRoot());
         if(workDataT.getData() == null)  workDataT.getLeft();
         workDataT.getLeft();
         captureS1=cap.CaptureTree(workDataT);
         workDataT.setCurrent(workDataT.getRoot());
         if(workDataT.getData() == null)
         workDataT.getLeft();
         workDataT.getLeft();
         workDataT=
         rc.ReplaceCapture(workDataT,searchS1,captureS1);
         workDataT.pretrav();
         workDataT.setCurrent(workDataT.getRoot());
         if(workDataT.getData() == null) workDataT.getLeft();
         workDataT.getLeft();
   }
     if(CheckQ(IDBranch,finalT) == true) {
          finalT=currentFinalT(finalTx, finalTy);
     if(workDataT.getQuantifier().equals("1")) {}
     else {
        var.addElement(inA);
        if(inB.length()!=0)
          var.addElement(inB);
      }

     st.STree(finalT,workDataT,inA,inB);
     finalT=doSTree1();
     finalT=doSTree2();
     finalT=doSTree3();
     finalT=doSTree4();
     finalT=doSTree5();
     finalT=currentFinalT(workx,worky);
     }
     else{
          display.setText("Incorrect place, try again");
          finalT=currentFinalT(workx,worky);
          if(finalT.get lClicked()==true) finalT.set lClicked();
          pickedworkxy=false;
          finalT=currentFinalT(finalTx, finalTy);repaint();

     }
   else{
     display.setText("Try again. Pick another variable.");
     if(finalT.getLClicked()==false) finalT.setLClicked();
       finalT.setLClosed();
```

```
        pickedworkxy=false;

      }
     }
     else st.STree(finalT,workDataT,"BLANK","BLANK");
     finalT=currentFinalT(workx,worky);
     workDataT.getParent();finalT.inDataTree(workDataT);
     finalT=doSTree1();
     finalT=doSTree2();
     finalT=doSTree3();
     finalT=doSTree4();
     finalT=doSTree5();
     inputA.setText("");
     inputB.setText("");
     workx=workx-workx;worky=worky-worky;
     finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
     mousex=mousex-mousex;mousey=mousey-mousey;repaint();
     pickedworkxy=false;
  }
  else {

      if(isCheckedOff(workDataT)==true) finalT.setCheckedOff();
      else finalT.setClicked();
      if(workDataT.getQuantifier() =="0") workDataT.getParent();
      if(workDataT.getQuantifier() != null){
         if(Character.isLowerCase(inputAA.charAt(0))||
            Character.isLowerCase(inputBB.charAt(0))||
            ){
            if(inB.length()==0){
               inB="BLANK";
             }
             else{
                searchS2=search.SearchTree(workDataT,inB);
                workDataT.setCurrent(workDataT.getRoot());
                if(workDataT.getData() == null) workDataT.getLeft();
                captureS2=cap.CaptureTree(workDataT);
                workDataT.setCurrent(workDataT.getRoot());
                if(workDataT.getData() == null) workDataT.getLeft();
                workDataT.getRight();
                workDataT=rc.ReplaceCapture(workDataT,searchS2,
                captureS2);
                workDataT.pretrav();
                workDataT.setCurrent(workDataT.getRoot());
                if(workDataT.getData() == null) workDataT.getLeft();
             }

         if(searchS1.equals(inA) ){
           display.setText(searchS1+"111!!!!Warning capture");
           workDataT.setCurrent(workDataT.getRoot());
           if(workDataT.getData() == null) workDataT.getLeft();
           captureS1=cap.CaptureTree(workDataT);
               workDataT.setCurrent(workDataT.getRoot());
           if(workDataT.getData() == null)
              workDataT.getLeft();
           workDataT.getRight();
           workDataT=rc.ReplaceCapture(workDataT,searchS1,captureS1);
           workDataT.pretrav();
           workDataT.setCurrent(workDataT.getRoot());
           if(workDataT.getData() == null) workDataT.getLeft();
         }

    if(CheckQ(IDBranch,finalT) == true) {
          finalT=currentFinalT(finalTx, finalTy);
          if(workDataT.getQuantifier().equals("1")) {}
           else {
              var.addElement(inA);
              if(inB.length()!=0)
```

```
                var.addElement(inB);
              }
            st.STree(finalT,workDataT,inA,inB);
            finalT=doSTree1();
            finalT=doSTree2();
            finalT=doSTree3();
            finalT=doSTree4();
            finalT=doSTree5();
            repaint();
          }
        else{
                display.setText("Incorrect place, try again");
                finalT=currentFinalT(workx,worky);
                if(finalT.getClicked()==true) finalT.setClicked();
                pickedworkxy=false;
                finalT=currentFinalT(finalTx, finalTy);repaint();

             }
          else{
            display.setText("Try again. Pick another variable.");
            if(finalT.getClicked()==false) finalT.setClicked();
              finalT.setClosed();
              pickedworkxy=false;

         }

        }
        else st.STree(finalT,workDataT,"BLANK","BLANK");repaint();
        finalT=currentFinalT(workx,worky);
         workDataT.getParent();finalT.inDataTree(workDataT);
         finalT=doSTree1();
         finalT=doSTree2();
         finalT=doSTree3();
         finalT=doSTree4();
         finalT=doSTree5();
         inputA.setText("");
      inputB.setText("");
      workx=workx-workx;worky=worky-worky;
      finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
      mousex=mousex-mousex;mousey=mousey-mousey;repaint();
      pickedworkxy=false;
     }
     if(getWorkBranch(getBranchxy(workx,worky),
        getBranch(workx,worky)).getQuantifier() != null){
        inA=inputA.getText();
     }
      repaint();
      inputA.setText("");
      inputB.setText("");
      workx=workx-workx;worky=worky-worky;
      finalTx=finalTx-finalTx;finalTy=finalTy-finalTy;
      mousex=mousex-mousex;mousey=mousey-mousey;
      pickedworkxy=false;
   }
   if(finalT.getRCheckedOff()==true && finalT.getLCheckedOff() ==true){
       if(finalT.getCurrent() == finalT.getBranch6())
         finalT.setBranch6(finalT.getRoot());
       else if(finalT.getCurrent() == finalT.getBranch7())
         finalT.setBranch7(finalT.getRoot());
       else{//do nothing}
     }
     repaint();
  }

catch(Exception e) {

}
```

```
            return true;
          }


public void paint( Graphics g) {

        int [] holdx = new int[30];
        int [] holdy = new int[30];
        boolean start=true;
        boolean loop = true;
        boolean right = false;
        boolean left = false;
        boolean parent = false;
        boolean noGoLeft=false;
        boolean prevLClosed=false;
        boolean prevRClosed=false;
        int count=1;
        int R=0;
        int L=0;
        int i=0;
        int level= (-1);
        boolean centerYes =true;
        int sx=400, sy=160 ,x=450,y=200;
        int sxA=0, sxB=0, syA=0, syB=0;
        int a =0; int b=0;
        int helploop=30;// to stop endless loops
        g.setFont(new Font( "CourierNew", Font.BOLD, 12 ) );
        finalT.setCurrent(drawT.getRoot());
          if(finalT.getBranch1().getDTestClose() !=
             null||finalT.getBranch1().getRTestClose() != null){
                    finalT.setCurrent(finalT.getBranch1());
                    finalT=closedT(finalT);
                  }
           if(finalT.getBranch2().getDTestClose() !=
              null||finalT.getBranch2().getRTestClose() != null){
                    finalT.setCurrent(finalT.getBranch2());
                    finalT=closedT(finalT);
                  }//display.setText("help2"+finalT.getData()+finalT.getClosed());
           if(finalT.getBranch3().getDTestClose() !=
              null||finalT.getBranch3().getRTestClose() != null){
                    finalT.setCurrent(finalT.getBranch3());
                    finalT=closedT(finalT);
                  }
           if(finalT.getBranch4().getDTestClose() !=
              null||finalT.getBranch4().getRTestClose() != null){
                    finalT.setCurrent(finalT.getBranch4());
                    finalT=closedT(finalT);
                  }
           if(finalT.getBranch5().getDTestClose() !=
              null||finalT.getBranch5().getRTestClose() != null){
                    finalT.setCurrent(finalT.getBranch5());
                    finalT=closedT(finalT);
                  }
        drawT = finalT ;
        drawT.setCurrent(drawT.getRoot());
        g.setColor(Color.black);

        if(start1 != ""){t1.setCurrent(t1.getRoot());
           if(t1.getClicked()==true) g.setColor(new Color(51,153,0));
           if(t1.getCheckedOff() ==true)
           g.setColor(new Color(102,102,102));
           g.drawString("1: "+start1,400,100);
           g.setColor(Color.black);
         }
        if(start2 != ""){t2.setCurrent(t2.getRoot());
```

```
    if(t2.getClicked()==true) g.setColor(new Color(51,153,0));
    if(t2.getCheckedOff() ==true)
    g.setColor(new Color(102,102,102));
    g.drawString("2: "+start2,400,110);
    g.setColor(Color.black);
}
if(start3 != ""){t3.setCurrent(t3.getRoot());
    if(t3.getClicked()==true) g.setColor(new Color(51,153,0));
    if(t3.getCheckedOff() ==true)
    g.setColor(new Color(102,102,102));
    g.drawString("3: "+start3,400,120);
    g.setColor(Color.black);
 }
g.drawLine(450,130,450,140);
  holdx[i]=400; holdy[i]=160 ;

if(drawT.getCurrent() == drawT.getRoot()){
    start=true;
    sx=holdx[i];
    sy=holdy[i];
    if(drawT.getCenter() == true){
      centerYes=true;
      }
    else centerYes=false;

          if(drawT.getData() != null) {
          if(drawT.getClicked()==true) g.setColor(new Color(51,153,0));
          if(drawT.getCheckedOff()==true) g.setColor(new Color(102,102,102));
          if(drawT.getCurrent().getClosed() == true) {g.setColor(Color.red);
            prevRClosed=true;}
          else prevRClosed=false;
          g.drawString(whichBranch(drawT)+drawT.getData(),sx,sy);
          g.setColor(Color.black);

          i=i+1;
          holdx[i]=sx; holdy[i]=sy ;

          Branchxy(whichBranch(drawT),sx,sy);
             if(drawT.getCenter() == true){
                 centerYes=true;
              }
             else
                 centerYes=false;
        }

      else if(drawT.getRightData() != null) {
             if(drawT.getLClicked()==true) g.setColor(new Color(51,153,0));
             if(drawT.getLCheckedOff()==true) g.setColor(new Color(102,102,102));
               //grey
             if(drawT.getClosed()==true||drawT.getCurrent().getRClosed() ==
                true||drawT.getCurrent().getLClosed() ==true){
                g.setColor(Color.red);
                prevRClosed=true;
             }
             else prevRClosed=false;
             slength = drawT.getLeftData().length();
             g.drawString(whichBranch(drawT)+drawT.getLeftData(),sx+20+20-
             (slength)*2,sy-10) ;
             g.setColor(Color.black);
             if(drawT.getRClicked()==true) g.setColor(new Color(51,153,0));
             if(drawT.getRCheckedOff()==true)
                g.setColor(new Color(102,102,102));//grey
             if(drawT.getClosed()==true||drawT.getCurrent().getRClosed() ==
                true||drawT.getCurrent().getLClosed() ==true)
                g.setColor(Color.red);
             LBranchxy(whichBranch(drawT),sx+20+20-(slength)*2,sy-10);
             g.drawString(drawT.getRightData(),sx+20+20-(slength)*2,sy+10) ;
```

```
            g.setColor(Color.black);

            RBranchxy(whichBranch(drawT),sx+20+20-(slength)*2,sy+10);

            i=i+1;
            holdx[i]=sx; holdy[i]=sy ;


             if(drawT.getCenter() == true){
               centerYes=true;
             }
             else
               centerYes=false;
       }
       else {}

}

 while(loop==true) {

       count=count+1;
       if(count > stoploop){
         loop=false;
         left=false;
         right=false;
         parent=false;
       }
      if(left == true) {

        right = false;
        left = true;
        bottomLeft=true;

      }
      else {
        if(moveRight(drawT) == true) right =true;
          else right = false;
      }

        while(right == true) {
          R+=1;
          drawT.getRight();
          level=level+1;
          sx=holdx[i];
          sy=holdy[i];
          R) ;
          if(centerYes ==true){
                if(sx==400){
                     if(drawT.getCurrent().getLineClosed()==true){
                         g.setColor(Color.red);
                      else g.setColor(Color.black);
                    g.drawLine(sx+50,sy+10,sx+50,sy+45);
                  }
                  else{
                    if(drawT.getCurrent().getLineClosed()==true){
                       g.setColor(Color.red);
                       else g.setColor(Color.black);
                     }
                    g.drawLine(sx+5,sy+10,sx+5,sy+45);
                   }
             if(drawT.getData() != null) {
             if(drawT.getClicked()==true) g.setColor(new Color(51,153,0));
                //green
             if(drawT.getCheckedOff()==true)
                g.setColor(new Color(102,102,102));//grey
             if(drawT.getCurrent().getClosed() == true) g.setColor(Color.red);
             if(sx==400){
```

```
    g.drawString(whichBranch(drawT)+drawT.getData(),sx+20+20-
      (slength)*2,sy+60) ;g.setColor(Color.black);
    Branchxy(whichBranch(drawT),sx+20+20-(slength)*2,sy+60);
}
else{
    g.drawString(sx+","+sy+","+whichBranch(drawT)+
    drawT.getData(),sx,sy+60) ; g.setColor(Color.black);
    Branchxy(whichBranch(drawT),sx,sy+60);
}
sx=sx; sy=sy+60;

i=i+1;
holdx[i]=sx; holdy[i]=sy ; //i=i+1;
   if(drawT.getCenter() == true){
     centerYes=true;
    }
   else
     centerYes=false;
}
else {
    if(sx==400){
          if(drawT.getCurrent().getLineClosed()==true)
              g.setColor(Color.red);
           else g.setColor(Color.black);

          g.drawLine(sx+50,sy+10,sx+50,sy+45);
       }
       else{
         if(drawT.getCurrent().getLineClosed()==true)
              g.setColor(Color.red);
           else g.setColor(Color.black);

         g.drawLine(sx+5,sy+10,sx+5,sy+45);
        }
     if(drawT.getLClicked()==true) g.setColor(new Color(51,153,0));
     if(drawT.getLCheckedOff()==true)
        g.setColor(new Color(102,102,102)); //grey
     if(drawT.getCurrent().getRClosed()
        ||drawT.getCurrent().getLClosed() == true|
        ||drawT.getClosed()==true){
        g.setColor(Color.red); prevRClosed=true;
      }

     if(sx==400){
       g.drawString(whichBranch(drawT)+drawT.getLeftData()
       ,sx+20+20-(slength)*2,sy-10+70) ;
       LBranchxy(whichBranch(drawT),sx+20+20-(slength)*2,sy-10+70);
       //g.setColor(Color.black);
      }
     else{
        g.drawString(sx+","+sy+","+whichBranch(drawT)+sx+"b"+sy+
        drawT.getLeftData(),sx,sy-10+70);
        //g.setColor(Color.black);
        LBranchxy(whichBranch(drawT),sx,sy-10+70);
       }

     g.setColor(Color.black);
     if(drawT.getRClicked()==true) g.setColor(new Color(51,153,0));
     if(drawT.getRCheckedOff()==true) g.setColor(new
        Color(102,102,102)); //grey
     if(drawT.getCurrent().getRClosed()
       ||drawT.getCurrent().getLClosed() == true
       ||drawT.getClosed()==true) g.setColor(Color.red);
     if(sx==400){
         g.drawString(drawT.getRightData(),sx+20+20-
        (slength)*2,sy+10+70) ;
         RBranchxy(whichBranch(drawT),sx+20+20-
```

```
            (slength)*2,sy+10+70);
          }
          else {
             g.drawString(whichBranch(drawT)+sx+","+
             sy+drawT.getRightData(),sx,sy+10+70) ;
             RBranchxy(whichBranch(drawT),sx,sy+10+70);
           }
         if(drawT.getRCheckedOff()==true)
             g.setColor(Color.black);
            sx=sx; sy=sy+70;

         i=i+1;
         holdx[i]=sx; holdy[i]=sy ; // i=i+1;
            if(drawT.getCenter() == true){
               centerYes=true;
             }
            else
               centerYes=false;
            }

  }
else{
    if(drawT.getData() != null) {
    if(drawT.getCurrent().getLineClosed()==true)  {
       g.setColor(Color.red);
     }
    else g.setColor(Color.black);
    g.drawLine(sx+50,sy+10,sx+200-(level*30),sy+30);
    g.setColor(Color.black);
    if(drawT.getClicked()==true) g.setColor(new Color(51,153,0));
    //green
    if(drawT.getCheckedOff()==true) g.setColor(new Color(102,102,102));
    //grey
    if(drawT.getClosed()==true) {g.setColor(Color.red);
       prevRClosed=true;}
    else{
        g.setColor(Color.black);
        prevRClosed=false;
    }
    g.drawString(drawT.getData(),sx+200-(level*30),sy+40) ;
    g.setColor(Color.black);
    sx=sx+200-(level*30); sy=sy+40;
    Branchxy(whichBranch(drawT),sx,sy);
    i=i+1;
    holdx[i]=sx; holdy[i]=sy ; //i=i+1;
        if(drawT.getCenter() == true){
          centerYes=true;
         }
        else
          centerYes=false;
    }

    else {
       if(drawT.getCurrent().getLineClosed()==true){
          g.setColor(Color.red);
       }
       else g.setColor(Color.black);
         g.drawLine(sx+50,sy+10,sx+10+200-(level*30),sy+20);
         if(drawT.getLClicked()==true) g.setColor(new Color(51,153,0));
         if(drawT.getLCheckedOff()==true)
            g.setColor(new Color(102,102,102)); //grey
         if(drawT.getCurrent().getRClosed() ==
             true||drawT.getCurrent().getLClosed() == true)
         g.setColor(Color.red);
         g.drawString(drawT.getLeftData(),sx+200-(level*30),sy-10+40) ;
         g.setColor(Color.black);
         LBranchxy(whichBranch(drawT),sx+200-(level*30),sy-10+40);
```

```
            if(drawT.getRClicked()==true) g.setColor(new Color(51,153,0));
            if(drawT.getRCheckedOff()==true) g.setColor(new
               Color(102,102,102));
            if(drawT.getCurrent().getRClosed() ==
               true||drawT.getCurrent().getLClosed() == true)
               g.setColor(Color.red);
            g.drawString(drawT.getRightData(),sx+200-(level*30),
            sy+10+40) ;
            g.setColor(Color.black);
            RBranchxy(whichBranch(drawT),sx+200-(level*30),sy+10+40);
            sx=sx+200-(level*10); sy=sy+40;
            i=i+1;
            holdx[i]=sx; holdy[i]=sy ;
                 if(drawT.getCenter() == true){
                   centerYes=true;
                  }
                 else
                   centerYes=false;
             }
        }
      if(moveRight(drawT) == true) {
        right = true;
       }
      else {
        right = false;
        loop = true;
        if(drawT.getClosed()==true || drawT.getRClosed()==true||
           drawT.getLClosed()==true)
           prevLClosed=true;
        else prevLClosed=false;
      }
    }
     if(bottomLeft==false){
       if(moveParent(drawT) == true){
           R=R-1;
           drawT.getParent();
           level=level-1;
           i=i-1;
           if(drawT.getCurrent() == drawT.getRoot())
                  start = false;
           }
       }
       else {
         bottomLeft=false;
       }
      if(moveLeft(drawT) ==true) {
        left = true ;
        while ( left == true ){
            L=L+1;
        sx = holdx[i];
        sy = holdy[i];
        sxA= sx+50;
        syB=sy+10 ;
        sxB= sx-200+75+(level*40);
        syB= sy+27;
        sx = sx-200;
       if(moveLeft(drawT)==true ) {
                drawT.getLeft();));
                level=level+1;
            }
          if(drawT.getData() != null) {
            if(drawT.getClicked()==true) g.setColor(new Color(51,153,0));
            if(drawT.getCheckedOff()==true)
               g.setColor(new Color(102,102,102)); //grey
            if(drawT.getClosed() == true) g.setColor(Color.red) ;
            else g.setColor(Color.black);
            g.drawString(drawT.getClosed()+whichBranch(drawT)+
```

```
    drawT.getData(),sx+10+(level*40)+30,sy) ;
    g.setColor(Color.black);
    if(drawT.getCurrent().getLineClosed()==true)
      g.setColor(Color.red);
      else g.setColor(Color.black);

  if(drawT.getClosed() ==true) prevLClosed=true;
  else prevLClosed = false;
  g.drawLine(sx+50+200,sy+10-40,sx-200+75+(level*40)+200,sy+27-40);
  sx=sx+10+(level*40)+30; sy=sy;
  Branchxy(whichBranch(drawT),sx,sy);
  i=i+1;
  holdx[i]=sx; holdy[i]=sy ;
      if(drawT.getCenter() == true){
        centerYes=true;
       }
  }

  else {
        if(drawT.getLClicked()==true)
           g.setColor(new Color(51,153,0));
        if(drawT.getCheckedOff()==true)
           g.setColor(new Color(102,102,102)); //grey
        if(drawT.getCurrent().getClosed() == true)
           g.setColor(Color.red);
        g.drawString(drawT.getLeftData(),sx+(level*40)+30,sy-10) ;
        LBranchxy(whichBranch(drawT),sx+(level*40)+30,sy-10);
        if(drawT.getRClicked()==true)
           g.setColor(new Color(51,153,0));
        if(drawT.getCheckedOff()==true)
           g.setColor(new Color(102,102,102)); //grey
        if(drawT.getCurrent().getClosed() == true)
           g.setColor(Color.red);
        g.drawString(drawT.getRightData(),sx+(level*40)+30,sy+10) ;
        RBranchxy(whichBranch(drawT),sx+(level*40)+30,sy+10);
        sx=sx+(level*40)+30; sy=sy+5;
        i=i+1;
        holdx[i]=sx; holdy[i]=sy ; //i=i+1;
            if(drawT.getCenter() == true){
              centerYes=true;
             }
        }
       if(moveRight(drawT) == true) {
           left = false;
           parent = false;
           loop = true;
           right = true;
   }
   else {
      if(moveParent(drawT) ==true){
          L=L-1;
          R=R-1;
          i=i-1;
          drawT.getParent();
          level=level-1;
          left =false;
          if(drawT.getCurrent() == drawT.getRoot())
             if( start==false ) {
               left =false;
               right=false;
               loop =false;
              } //start
            } // root
          else{

if(moveParent(drawT) == true) {
                  R=R-1;
```

```
                i=i-1;
                drawT.getParent();
                level=level-1;
                parent=true;

            }
            while( parent == true){
                if(drawT.getCurrent() == drawT.getRoot();
                    if(moveLeft(drawT) == true){
                      if(start==true){
                        parent = false;
                        loop =true;
                        left=true;
                        right=false;
                        start=false;
                       }
                      else{
                        parent = false;
                        loop =false;
                        left= false;
                        right=false;
                      }
                    }
                    else {
                        loop=false;
                    }
                }
                else{
                     if(moveLeft(drawT) == true){
                       if(L <=  1){
                          loop =true;
                          left=true;
                          parent=false;
                          right=false;
                        }
                        else{
                           if(moveParent(drawT)){
                            parent = true;
                            drawT.getParent();
                            level=level-1;
                            i=i-1;
                            R=R-1;
                            L=L-1;
                          }
                        }
                    }
                     //*/
                    else {
                       if(moveParent(drawT)){
                           parent = true;
                           drawT.getParent();
                           level=level-1;
                           i=i-1;
                           R=R-1;
                         }
                       else {
                           parent = false;
                           left = false;
                           right = false;
                           loop = false;
                       }
                    }
                }
        }


    }
```

```
        }

    }

 }
} // getParent()
    else{
        if(moveParent(drawT) ==true){
           R=R-1;
           i=i-1;
           drawT.getParent(); travelLeft=false;
           level=level-1;
          }
           if(drawT.getCurrent() == drawT.getRoot()){
              if(start == false){
                right=false;
                parent=false;
                left=false;
                loop = false;
              }
              else {
               start = false;
               if(moveLeft(drawT) == true){
                  left=true;
                  loop=true;
                  if(drawT.getCurrent() == drawT.getRoot()){
                      if(start == false){
                      }
                     }
                   }
               }
               else {
                  loop=false;
                }
              }
            }
           else{
               if(moveParent(drawT) ==true){
                 if(travelLeft==true){
                    R=R-1;
                    i=i-1;
                    drawT.getParent();
                    level=level-1;
                    travelLeft=false;
                 }
                  else {
                    travelLeft=false;
                    if(moveLeft(drawT)==true){
                        if(moveParent(drawT)==true){
                            if(start==false && L==2 && level==1){
                                loop =true;
                                left=false;
                                parent=true;
                                right=false;
                             }
                             else{
                               loop =true;
                               left=true;
                               parent=false;
                               right=false;
                             }
                         }
                         else{
                              loop =true;
                              left=true;
                              parent=false;
                              right=false;
```

```
                        }

                    }
                  }
                   parent = true;
                }
               while( parent == true ){
                      if(drawT.getCurrent() == drawT.getRoot()){
                        if(start == false){
                           parent=false;
                           loop=false;
                           right=false;
                           left=false;
                        }
                        else {
                        start = false;
                          if(moveLeft(drawT) == true){
                            parent = false;
                            loop =true;
                            left=true;
                          }
                          else {
                            loop=false;
                          }
                        }
                       }
                       else{
                            if(moveLeft(drawT) == true){
                              if(L < 1|| L==level||L-level==2){
                                 loop =true;
                                 left=true;
                                 right=false;
                                 parent=false;
                               }

                                 else{
                                    if(moveParent(drawT)){
                                     parent = true;
                                     i=i-1;
                                     R=R-1;
                                     drawT.getParent();
                                     level=level-1;
                                   }
                                  }
                              }
                              else {
                                 if(moveParent(drawT)){
                                     parent = true;
                                     i=i-1;
                                     R=R-1;
                                     drawT.getParent();
                                     level=level-1;
                                   }
                                 else {
                                     parent = false;
                                     left = false;
                                     right = false;
                                     loop = false;
                                 }
                              }
                       }
                  }
            }// else

     } //else
if(count >stoploop){
loop=false;
```

```
                left=false;
                right=false;
                parent=false;
              } //***********
             //loop =false;
            }

     } //end paint
public boolean moveRight(BinaryTree mv) {
 if( mv.getCurrent().getRight() != null)
    return true;
  else return false;
}

public boolean moveLeft(BinaryTree mv) {
if( mv.getCurrent().getLeft() != null)
    return true;
  else return false;
}

public boolean moveParent(BinaryTree mv) {
if( mv.getCurrent().getParent() != null)
    return true;
  else return false;
}

public BinaryTree closedT(BinaryTree d){
     if(d.getCurrent() == d.getRoot()) return d;
     hf=d.getCurrent();System.out.println( d.getClosed()+
     Cloop=true;
     Cloop2=true;
     if(d.getDTestClose() != null){
        pred=d.getDTestClose(); closedNot=d.getNot();
        d.getParent();
        while(Cloop == true){
          if(d.getDTestClose() != null){
             if(d.getDTestClose().equals(pred)){
                if(d.getNot()!=closedNot){
                   finish=d.getCurrent();
                   d.setCurrent(hf);
                   d.setClosed();d.setRClosed(); d.setLClosed();
                      while(Cloop2==true){
                        if(finish == d.getCurrent()){
                           Cloop2 =false;
                           d.setClosed();d.setRClosed(); d.setLClosed();
                        }
                        else {
                          if(moveRight(d)==false)
                              d.getCurrent().setLineClosed();
                          d.getParent();
                        }
                      } //while
                   return d;
                }//if

            } //if
             if(d.getRoot() == d.getCurrent()) Cloop=false;
             d.getParent();
          }
          else{
             if(d.getRTestClose() != null){
                if(d.getRTestClose().equals(pred)){
                if(d.getRnot()!=closedNot){
                   finish=d.getCurrent();
                   d.setCurrent(hf);
                   d.setClosed();d.setRClosed(); d.setLClosed();
                      while(Cloop2 == true){
```

```
                    if(finish == d.getCurrent()){
                      d.setClosed();d.setRClosed(); d.setLClosed();
                      Cloop2 = false;
                    }
                    else{
                       if(moveRight(d)==false)
                          d.getCurrent().setLineClosed();
                       d.getParent();
                     }
                  } //while
               return d;
              }//if

        }
            if(d.getRoot() == d.getCurrent()) Cloop=false;
         }
         if(d.getLTestClose() != null){
            if(d.getLTestClose().equals(pred)){
             if(d.getLnot()!=closedNot){
                finish =d.getCurrent();
                d.setCurrent(hf);
                d.setClosed();d.setRClosed(); d.setLClosed();
                   while(Cloop2==true){
                      if(finish == d.getCurrent()){
                       Cloop2=false;
                       d.setClosed();d.setRClosed(); d.setLClosed();
                      }
                      else {
                       if(moveRight(d)==false)
                            d.getCurrent().setLineClosed();
                        d.getParent();
                      }
                    } //while

              }//if

        } //if
            if(d.getRoot() == d.getCurrent()){
              Cloop=false;
              return d;
             }
            d.getParent();
         }
         else{
              if(d.getRoot() == d.getCurrent()){
              Cloop=false;
              return d;
             }
            d.getParent();
         }
      }
   } //while
}
else if(d.getRTestClose()!= null){
      pred=d.getRTestClose(); closedNot=d.getRnot();
      d.getParent();
      while(Cloop == true){
        if(d.getDTestClose() != null){
          if(d.getDTestClose().equals(pred)){
            if(d.getNot()!=closedNot){
               finish=d.getCurrent();
               d.setCurrent(hf);
               d.setClosed();d.setRClosed(); d.setLClosed();
                  while(Cloop2==true){
                    if(finish==d.getCurrent()){
                      Cloop2=false;
                      d.setClosed();d.setRClosed(); d.setLClosed();
```

```
            }
            else {
              if(moveRight(d)==false)
                  d.getCurrent().setLineClosed();
              d.getParent();
            }
          } //while
          return d;
        }//if
 //else d.getParent();
} //if
 if(d.getRoot() == d.getCurrent()) Cloop=false;
 d.getParent();
}
else{
 if(d.getRTestClose() != null){
   if(d.getRTestClose().equals(pred)){
      if(d.getRnot()!=closedNo t){
        finish = d.getCurrent();
        d.setCurrent(hf);
        d.setClosed();d.setRClosed(); d.setLClosed();
          while(Cloop2 == true){
            if(finish == d.getCurrent()) {
               Cloop2=false;
               d.setClosed();d.setRClosed(); d.setLClosed();
             }
            else{
               d.getParent();
               if(moveRight(d)==false)
                   d.getCurrent().setLineClosed();
             }
          } //while

      }//if

} //if
    if(d.getRoot() == d.getCurrent()) Cloop=false;
 }
 if(d.getLTestClose() != null){

    if(d.getRoot() == d.getCurrent()){
      if(d.getLTestClose().equals(pred)){
        if(d.getLnot()!=closedNot){
           finish = d.getCurrent();
           d.setCurrent(hf);
           d.setClosed();d.setRClosed(); d.setLClosed();
           while(Cloop2==true){
             if(finish == d.getCurrent()){
                Cloop2=false;
                d.setClosed();d.setRClosed(); d.setLClosed();
             }
             else{
               if(moveRight(d)==false)
                  d.getCurrent().setLineClosed();
               d.getParent();
             }
           } //while

      }//if

} //if
      Cloop=false;
      d.setCurrent(hf);
     }
  d.getParent();
 }
  else{
```

```
        if(d.getRoot() == d.getCurrent()){
           Cloop=false;
        }
       }
        else d.getParent();
    }
 }
}//while
 d.setCurrent(hf);
 pred=d.getLTestClose(); closedNot=d.getLnot();
 d.getParent();
 Cloop=true;
 while(Cloop == true){
   if(d.getDTestClose() != null){
     if(d.getDTestClose().equals(pred)){
       if(d.getNot()!=closedNot){
         finish=d.getCurrent();
         d.setCurrent(hf);
         d.setClosed();d.setRClosed(); d.setLClosed();
         while(Cloop2==true){
           if(finish == d.getCurrent()){
              Cloop2=false;
              d.setClosed();d.setRClosed(); d.setLClosed();
           }
           else {
              d.getParent();
              if(moveRight(d)==false)
                    d.getCurrent().setLineClosed();
            }
           } //while
      //else return d;
       }//if

  } //if
   if(d.getRoot() == d.getCurrent())
     Cloop=false;
     else
     d.getParent();
}
else{
   if(d.getRTestClose() != null){
     if(d.getRTestClose().equals(pred)){
       if(d.getRnot()!=closedNot){
         finish=d.getCurrent();
         d.setCurrent(hf);
         d.setClosed();d.setRClosed(); d.setLClosed();
         while(Cloop2==true){
           if(finish==d.getCurrent()){
              Cloop2=false;
              d.setClosed();d.setRClosed(); d.setLClosed();
           }
           else {
              d.getParent();
              if(moveRight(d)==false)
                    d.getCurrent().setLineClosed();

            }
           } //while

       }//if

  }
      if(d.getRoot() == d.getCurrent()) Cloop=false;
   }
   if(d.getLTestClose() != null){
     if(d.getLTestClose().equals(pred)){
       if(d.getLnot()!=closedNot){
```

```
                    finish=d.getCurrent();
                    d.setCurrent(hf);
                    d.setClosed();d.setRClosed(); d.setLClosed();
                       while(Cloop2==true){
                         if(finish ==d.getCurrent()){
                            Cloop2=false;
                            d.setClosed();d.setRClosed(); d.setLClosed();
                         }
                         else {
                            if(moveRight(d)==false)
                               d.getCurrent().setLineClosed();
                            d.getParent();
                          }
                       } //while

                   }//if

              } //if
                  if(d.getRoot() == d.getCurrent()){
                    Cloop=false;
                     d.setCurrent(hf);
                     return d;
                   }
                  d.getParent();
               }
                else{
                    if(d.getRoot() == d.getCurrent()){
                    Cloop=false;

                   }
                  if(d.getCurrent() == d.getRoot())
                     return d;
                  else
                     d.getParent();
               }
             }

          }//while
      }//RTestClose()
      else if(d.getLTestClose()!= null){
            while(Cloop == true){
            }
      }
      else{
       return d;
      }//LTestClose()
   return d;
}
public boolean CheckQ(String b,BinaryTree d){
   ChildNode homeO = new ChildNode();
   String branchID = b;
   homeO=d.getCurrent();
   while(d.getCurrent() != d.getRoot()){
        if(whichBranch(d).equals(b)){
           d.setCurrent(homeO);
           return true;
        }
        else   d.getParent();
    }
    d.setCurrent(homeO);
    return false;
}

public boolean checkOne(String b,BinaryTree d){
     int i=0;
   if(d.getQuantifier().equals("1")){
     while(i<var.size()){
```

```
      String s = (String)var.elementAt(i);
      if(s.equals(b))
         return true;
      else i++;
      }
        return false;
    }
    else return false;
}
public String whichBranch(BinaryTree d){
     if(d.getCurrent() == d.getBranch6())
        return "Branch 6";
     else if(d.getCurrent() == d.getBranch7())
        return "Branch 7";
     else if (d.getCurrent() == d.getBranch8())
        return "Branch 8";
     else if(d.getCurrent() == d.getBranch9())
        return "Branch 9";
     else if (d.getCurrent() == d.getBranch10())
        return "Branch 10";
     else if(d.getCurrent() == d.getBranch11())
        return "Branch 11";
     else if(d.getCurrent() == d.getBranch1())
        return "Branch 1";
     else if (d.getCurrent() == d.getBranch2())
        return "Branch 2";
     else if (d.getCurrent() == d.getBranch3())
        return "Branch 3";
     else if (d.getCurrent() == d.getBranch4())
        return "Branch 4";
     else if(d.getCurrent() == d.getBranch5())
        return "Branch 5";
     else return "blank";
 }

public void sendBranchBack(BinaryTree d){
     ChildNode homec = new ChildNode();
     homec=d.getCurrent();
     if(d.getRoot() != homec){
        d.getParent();
        if(d.getCurrent() == d.getBranch1()){
            d.setBranch1(d.getRoot());
        }
        else if (d.getCurrent() == d.getBranch2()){
            d.setBranch2(d.getRoot());
        }
        else if (d.getCurrent() == d.getBranch3()){
            d.setBranch3(d.getRoot());
        }
        else if (d.getCurrent() == d.getBranch4()){
            d.setBranch4(d.getRoot());
        }
        else{
            d.setBranch5(d.getRoot());System.out.println("branch5~~~~senback");
            bl5x=bl5x-bl5x;br5y=br5y-br5y;
        }
            d.setParent(homec);
      }//if home
}

public void Branchxy( String b, int x, int y){
     if( b.equals("Branch 1")){
         b1x = x; b1y=y ;
     }
     else if (b.equals("Branch 2")) {
        b2x= x; b2y =y;
     }
```

```
    else if (b.equals("Branch 3")) {
       b3x= x; b3y=y;
    }
    else if (b.equals("Branch 4")) {
       b4x= x; b4y=y;
    }
    else if (b.equals("Branch 5")) {
       b5x= x; b5y=y;
    }
    else if (b.equals("Branch 6")){
       b6x=x; b6y=y;
    }
    else if (b.equals("Branch 7")) {
       b7x= x; b7y=y;
    }
    else if (b.equals("Branch 8")){
       b8x=x; b8y=y;
    }
    else if (b.equals("Branch 9")){
       b9x=x; b9y=y;
    }
    else if (b.equals("Branch 10")) {
       b10x= x; b10y=y;
    }
    else if (b.equals("Branch 11")){
       b11x=x; b11y=y;
    }
    else {}
}

public void LBranchxy( String b, int x, int y){
    if( b.equals("Branch 1")){
        bl1x = x; bl1y=y ;
    }
    else if (b.equals("Branch 2")) {
       bl2x= x; bl2y =y;
    }
    else if (b.equals("Branch 3")) {
       bl3x= x; bl3y=y;
    }
    else if (b.equals("Branch 4")) {
       bl4x= x; bl4y=y;
    }
    else if (b.equals("Branch 5")) {
       bl5x= x; bl5y=y;
    }
    else if (b.equals("Branch 6")) {
       bl6x=x;bl6y=y;
    }
    else if (b.equals("Branch 7")) {
       bl7x= x; bl7y=y;
    }
    else if (b.equals("Branch 8")) {
       bl8x=x;bl8y=y;
    }
    else if (b.equals("Branch 9")) {
       bl9x=x;bl9y=y;
    }
    else if (b.equals("Branch 10")) {
       bl10x= x; bl10y=y;
    }
    else if (b.equals("Branch 11")) {
       bl11x=x;bl11y=y;
    }
    else{}
}
```

```
public void RBranchxy( String b, int x, int y){
     if(b.equals("Branch 1") ){
         br1x = x; br1y=y ;
     }
     else if (b.equals("Branch 2")) {
        br2x= x; br2y =y;
     }
     else if (b.equals("Branch 3")) {
        br3x= x; br3y=y;
     }
     else if (b.equals("Branch 4")) {
        br4x= x; br4y=y;
     }
     else if (b.equals("Branch 5")) {
        br5x= x; br5y=y;
     }
     else if (b.equals("Branch 6")){
        br6x=x;br6y=y;
     }
     else if (b.equals("Branch 7")) {
        br7x= x; br7y=y;
     }
     else if (b.equals("Branch 8")){
        br8x=x;br8y=y;
     }
     else if (b.equals("Branch 9")){
        br9x=x;br9y=y;
     }
     else if (b.equals("Branch 10")) {
        br10x= x; br10y=y;
     }
     else if (b.equals("Branch 11")){
        br11x=x;br11y=y;
     }
     else{}
 }


public BinaryTree getWorkBranch( String a, BinaryTree c){
       if(a.equals(c.getBranch1().getData())){
          c.setCurrent(c.getBranch1());
          return c ;
       }
       else if(a.equals(c.getBranch2().getData())){
          c.setCurrent(c.getBranch2());
          return c ;
       }
       else if(a.equals(c.getBranch3().getData())){
          c.setCurrent(c.getBranch3());
          return c ;
       }
       else if(a.equals(c.getBranch4().getData())){
          c.setCurrent(c.getBranch4());
          return c ;
       }
       else if(a.equals(c.getBranch5().getData())){
          c.setCurrent(c.getBranch5());
          return c ;
       }
       else if(a.equals(c.getBranch6().getData())){
          c.setCurrent(c.getBranch6());
          return c ;
       }
       else if(a.equals(c.getBranch7().getData())){
          c.setCurrent(c.getBranch7());
          return c ;
       }
```

```
       else if(a.equals(c.getBranch8().getData())){
          c.setCurrent(c.getBranch8());
          return c ;
       }
       else if(a.equals(c.getBranch9().getData())){
          c.setCurrent(c.getBranch9());
          return c ;
       }
       else if(a.equals(c.getBranch10().getData())){
          c.setCurrent(c.getBranch10());
          return c ;
       }
       else {
          c.setCurrent(c.getBranch11());
          return c ;
       }

 }//
public boolean isCheckedOff(BinaryTree d){
      if(d.getQuantifier() != null) {//\u2200='For All' operator
         if("\u2200".equals(d.getQuantifier()))
            if(d.getNot() == true){
              if(d.getCheckedOff()==false)
                 //d.setCheckedOff();
                 return true;
            }
         if("\u2203".equals(d.getQuantifier())) //\u2203='There exists' operator
            if(d.getNot() == false){
               if(d.getCheckedOff()==false)
                 //d.setCheckedOff();
                 return true;
             }
         }
       else if(d.getOperator() != null) {
              if(d.getCurrent().getCheckedOff()==false)
                 //d.getCurrent().setCheckedOff();
                 return true;
       }
       return false;
   }//end isCheckedOff

public BinaryTree doSTree1(){
      if(finalT.getBranch1() != finalT.getRoot()){
          finalT.setCurrent(finalT.getBranch1());
          if(moveRight(finalT)==false){
            if(finalT.getNodeTree() != null) {
             if(getWorkBranch(finalT.getBranch1().getNodeTree(),
                 finalT.getBranch1().getDataTree()).getOperator() != null){
                 st.STree(finalT,getWorkBranch(finalT.getBranch1().getNodeTree(),
                 finalT.getBranch1().getDataTree()),"*","*");
                return finalT;
              }
               else
                 return finalT;
            }
             else
               return finalT;
          }
      }
        else
          return finalT;
 }

public BinaryTree doSTree2(){
     if(finalT.getBranch2() != finalT.getRoot()){
```

```
            finalT.setCurrent(finalT.getBranch2());
            if(moveRight(finalT)==false){
              if(finalT.getNodeTree() != null) {
               if(getWorkBranch(finalT.getBranch2().getNodeTree(),
                  finalT.getBranch2().getDataTree()).getOperator() != null){
                  finalT.setCurrent(finalT.getBranch2());
                  st.STree(finalT,getWorkBranch(finalT.getBranch2().getNodeTree(),
                  finalT.getBranch2().getDataTree()),"*","*");
                  return finalT;
                }
                 else
                  return finalT;
              }
               else
                 return finalT;
            }
             else
              return finalT;
        }
         else
           return finalT;
 }

public BinaryTree doSTree3(){
     if(finalT.getBranch3() != finalT.getRoot()){
          finalT.setCurrent(finalT.getBranch3());
          if(moveRight(finalT)==false){
            if(finalT.getNodeTree() != null) {
             if(getWorkBranch(finalT.getBranch3().getNodeTree(),
                 finalT.getBranch3().getDataTree()).getOperator() != null){
                finalT.setCurrent(finalT.getBranch3());
                st.STree(finalT,getWorkBranch(finalT.getBranch3().getNodeTree(),
                finalT.getBranch3().getDataTree()),"*","*");
                return finalT;
              }
               else
                return finalT;
            }
             else
               return finalT;
          }
           else
            return finalT;
      }
       else
         return finalT;
 }

public BinaryTree doSTree4(){
     if(finalT.getBranch4() != finalT.getRoot()){
          finalT.setCurrent(finalT.getBranch4());
          if(moveRight(finalT)==false){
            if(finalT.getNodeTree() != null) {
             if(getWorkBranch(finalT.getBranch4().getNodeTree(),
                finalT.getBranch4().getDataTree()).getOperator() != null){
                finalT.setCurrent(finalT.getBranch4());
                st.STree(finalT,getWorkBranch(finalT.getBranch4().getNodeTree(),
                finalT.getBranch4().getDataTree()),"*","*");
                return finalT;
              }
               else
                return finalT;
            }
             else
               return finalT;
          }
           else
```

```
            return finalT;
        }
         else
           return finalT;
 }

public BinaryTree doSTree5(){
      if(finalT.getBranch5() != finalT.getRoot()){
           finalT.setCurrent(finalT.getBranch5());
          if(moveRight(finalT)==false){
            if(finalT.getNodeTree() != null){
             if(getWorkBranch(finalT.getBranch5().getNodeTree(),
                finalT.getBranch5().getDataTree()).getOperator() != null){
                finalT.setCurrent(finalT.getBranch5());
                st.STree(finalT,getWorkBranch(finalT.getBranch5().getNodeTree(),
                finalT.getBranch5().getDataTree()),"*","*");
                return finalT;
              }
               else
                return finalT;
            }
             else
              return finalT;
          }
           else
            return finalT;
      }
      else
        return finalT;

}//doSTree5

public String getBranchxy(int mx,int my) {
        if( (b1x+70 > mx && mx > b1x-3) &&
          (b1y+5 > my && my > b1y-10) ) {
           return finalT.getBranch1().getNodeTree();
          }
        else if( (bl1x+70 > mx && mx > bl1x-3) &&
          (bl1y+5 > my && my > bl1y-10) ) {
           return finalT.getBranch1().getLeftTree();
          }
          else if( (br1x+70 > mx && mx > br1x-3) &&
          (br1y+5 > my && my > br1y-10) ) {
           return finalT.getBranch1().getRightTree();
          }
    //.........
          else if( (b2x+70 > mx && mx > b2x-3) &&
          (b2y+5 > my && my > b2y-10) ) {
           return finalT.getBranch2().getNodeTree();
          }
        else if( (bl2x+70 > mx && mx > bl2x-3) &&
          (bl2y+5 > my && my > bl2y-10) ) {
           return finalT.getBranch2().getLeftTree();
          }
          else if( (br2x+70 > mx && mx > br2x-3) &&
          (br1y+5 > my && my > br1y-10) ) {
           return finalT.getBranch2().getRightTree();
          }
  //............
          else if( (b3x+70 > mx && mx > b3x-3) &&
          (b3y+5 > my && my > b3y-10) ) {
           return finalT.getBranch3().getNodeTree();
          }
        else if( (bl3x+70 > mx && mx > bl3x-3) &&
          (bl3y+5 > my && my > bl3y-10) ) {
           return finalT.getBranch3().getLeftTree();
          }
```

```
        else if( (br3x+70 > mx && mx > br3x-3) &&
        (br3y+5 > my && my > br3y-10) ) {
         return finalT.getBranch3().getRightTree();
        }
  //..............
        else if( (b4x+70 > mx && mx > b4x-3) &&
        (b4y+5 > my && my > b4y-10) ) {

         return finalT.getBranch4().getNodeTree();
        }
       else if( (bl4x+70 > mx && mx > bl4x-3) &&
         (bl4y+5 > my && my > bl4y-10) ) {
         return finalT.getBranch4().getLeftTree();
        }
        else if( (br4x+70 > mx && mx > br4x-3) &&
        (br4y+5 > my && my > br4y-10) ) {
         return finalT.getBranch4().getRightTree();
        }
  //..............
       else if( (b5x+70 > mx && mx > b5x-3) &&
         (b5y+5 > my && my > b5y-10) ) {
         return finalT.getBranch5().getNodeTree();
        }
       else if( (bl5x+70 > mx && mx > bl5x-3) &&
         (bl5y+5 > my && my > bl5y-10) ) {
         return finalT.getBranch5().getLeftTree();
        }
        else if( (br5x+70 > mx && mx > br5x-3) &&
        (br5y+5 > my && my > br5y-10) ) {
         return finalT.getBranch5().getRightTree();
        }
//.......................
        else if( (b6x+70 > mx && mx > b6x-3) &&
        (b6y+5 > my && my > b6y-10) ) {
         return finalT.getBranch6().getNodeTree();
        }
       else if( (bl6x+70 > mx && mx > bl6x-3) &&
         (bl6y+5 > my && my > bl6y-10) ) {
         return finalT.getBranch6().getLeftTree();
        }
        else if( (br6x+70 > mx && mx > br6x-3) &&
        (br6y+5 > my && my > br6y-10) ) {
         return finalT.getBranch6().getRightTree();
        }
 //..............
       else if( (b7x+70 > mx && mx > b7x-3) &&
         (b7y+5 > my && my > b7y-10) ) {

         return finalT.getBranch7().getNodeTree();
        }
       else if( (bl7x+70 > mx && mx > bl7x-3) &&
         (bl7y+5 > my && my > bl7y-10) )
         return finalT.getBranch7().getLeftTree();
        }
        else if( (br7x+70 > mx && mx > br7x-3) &&
        (br7y+5 > my && my > br7y-10) ) {
         return finalT.getBranch7().getRightTree();
        }
 //.......................
       else if( (b8x+70 > mx && mx > b8x-3) &&
        (b8y+5 > my && my > b8y-10) ) {
         return finalT.getBranch8().getNodeTree();
        }
       else if( (bl8x+70 > mx && mx > bl8x-3) &&
         (bl8y+5 > my && my > bl8y-10) ) {
         return finalT.getBranch8().getLeftTree();
        }
```

```
        else if( (br8x+70 > mx && mx > br8x-3) &&
        (br8y+5 > my && my > br8y-10) ) {
         return finalT.getBranch8().getRightTree();
        }
 //.................
       else if( (b9x+70 > mx && mx > b9x-3) &&
        (b9y+5 > my && my > b9y-10) ) {
         return finalT.getBranch9().getNodeTree();
        }
       else if( (bl9x+70 > mx && mx > bl9x-3) &&
        (bl9y+5 > my && my > bl9y-10) ) {
         return finalT.getBranch9().getLeftTree();
        }
        else if( (br9x+70 > mx && mx > br9x-3) &&
        (br9y+5 > my && my > br9y-10) ) {
         return finalT.getBranch9().getRightTree();
       }
//.................
       else if( (b10x+70 > mx && mx > b10x-3) &&
        (b10y+5 > my && my > b10y-10) ) {
         return finalT.getBranch10().getNodeTree();
        }
       else if( (bl10x+70 > mx && mx > bl10x-3) &&
        (bl10y+5 > my && my > bl10y-10) ) {
         return finalT.getBranch10().getLeftTree();
        }
        else if( (br10x+70 > mx && mx > br10x-3) &&
        (br10y+5 > my && my > br10y-10) ) {
         return finalT.getBranch10().getRightTree();
        }
 //.........................
       else if( (b11x+70 > mx && mx > b11x-3) &&
        (b11y+5 > my && my > b11y-10) ) {
         return finalT.getBranch11().getNodeTree();
        }
       else if( (bl11x+70 > mx && mx > bl11x-3) &&
        (bl11y+5 > my && my > bl1y-10) ) {
         return finalT.getBranch11().getLeftTree();
        }
        else if( (br11x+70 > mx && mx > br11x-3) &&
        (br11y+5 > my && my > br11y-10) ) {
         return finalT.getBranch11().getRightTree();
        }
 //.........................


    else
          return "error";
}

public BinaryTree getBranch(int mx,int my) {
       if( (b6x+70 > mx && mx > b6x-3) &&
        (b6y+5 > my && my > b6y-10) ) {
         return finalT.getBranch6().getDataTree();
        }
      else if( (bl6x+70 > mx && mx > bl6x-3) &&
        (bl6y+5 > my && my > bl6y-10) ) {
         return finalT.getBranch6().getRightDataTree();
        }
      else if( (br6x+70 > mx && mx > br6x-3) &&
        (br6y+5 > my && my > br6y-10) ) {
         return finalT.getBranch6().getLeftDataTree();
        }
//.....................
       else if( (b7x+70 > mx && mx > b7x-3) &&
        (b7y+5 > my && my > b7y-10) ) {
```

```
          return finalT.getBranch7().getDataTree();
        }
      else if( (bl7x+70 > mx && mx > bl7x-3) &&
        (bl7y+5 > my && my > bl7y-10) ) {
         return finalT.getBranch7().getRightDataTree();
        }
      else if( (br7x+70 > mx && mx > br7x-3) &&
        (br7y+5 > my && my > br7y-10) ) {
         return finalT.getBranch7().getLeftDataTree();
        }
//.....................
      else if( (b8x+70 > mx && mx > b8x-3) &&
        (b8y+5 > my && my > b8y-10) ) {
         return finalT.getBranch8().getDataTree();
        }
      else if( (bl8x+70 > mx && mx > bl8x-3) &&
        (bl8y+5 > my && my > bl8y-10) ) {
         return finalT.getBranch8().getRightDataTree();
        }
      else if( (br8x+70 > mx && mx > br8x-3) &&
        (br8y+5 > my && my > br8y-10) ) {
         return finalT.getBranch8().getLeftDataTree();
        }
//.....................
      if( (b9x+70 > mx && mx > b9x-3) &&
        (b6y+5 > my && my > b6y-10) ) {
         return finalT.getBranch9().getDataTree();
        }
      else if( (bl9x+70 > mx && mx > bl9x-3) &&
        (bl9y+5 > my && my > bl9y-10) ) {
         return finalT.getBranch9().getRightDataTree();
        }
      else if( (br9x+70 > mx && mx > br9x-3) &&
        (br9y+5 > my && my > br9y-10) ) {
         return finalT.getBranch9().getLeftDataTree();
        }
//.....................
      else if( (b10x+70 > mx && mx > b10x-3) &&
        (b10y+5 > my && my > b10y-10) ) {
         return finalT.getBranch10().getDataTree();
        }
      else if( (bl10x+70 > mx && mx > bl10x-3) &&
        (bl10y+5 > my && my > bl10y-10) ) {
         return finalT.getBranch10().getRightDataTree();
        }
      else if( (br10x+70 > mx && mx > br10x-3) &&
        (br10y+5 > my && my > br10y-10) )
         return finalT.getBranch10().getLeftDataTree();
        }
//.....................
      else if( (b11x+70 > mx && mx > b11x-3) &&
        (b11y+5 > my && my > b11y-10) ) {
         return finalT.getBranch11().getDataTree();
        }
      else if( (bl11x+70 > mx && mx > bl11x-3) &&
        (bl11y+5 > my && my > bl11y-10) ) {
         return finalT.getBranch11().getRightDataTree();
        }
      else if( (br11x+70 > mx && mx > br11x-3) &&
        (br11y+5 > my && my > br11y-10) ) {
         return finalT.getBranch11().getLeftDataTree();
        }
//.....................
      else if( (b1x+70 > mx && mx > b1x-3) &&
        (b1y+5 >my && my > b1y-10) ) {
         return finalT.getBranch1().getDataTree();
        }
```

```
      else if( (bl1x+70 > mx && mx > bl1x-3) &&
        (bl1y+5 > my && my > bl1y-10)) {
         return finalT.getBranch1().getRightDataTree();

        }
      else if( (br1x+70 > mx && mx > br1x-3) &&
        (br1y+5 > my && my > br1y-10) ) {
         return finalT.getBranch1().getLeftDataTree();
           //return helpfinal;
        }
//.................
      else if( (b2x+70 > mx && mx > b2x-3) &&
        (b2y+5 > my && my > b2y-10) ) {
         return finalT.getBranch2().getDataTree();
        }
      else if( (bl2x+70 > mx && mx > bl2x-3) &&
        (bl2y+5 > my && my > bl2y-10) ) {
         return finalT.getBranch2().getRightDataTree();
        }
      else if( (br2x+70 > mx && mx > br2x-3) &&
        (br2y+5 > my && my > br2y-10) ) {
         return finalT.getBranch2().getLeftDataTree();
        }
// .................
      else if( (b3x+70 > mx && mx > b3x-3) &&
        (b3y+5 > my && my > b3y-10) ) {
         return finalT.getBranch3().getDataTree();
        }
      else if( (bl3x+70 > mx && mx > bl3x-3) &&
        (bl3y+5 > my && my > bl3y-10) ) {
         return finalT.getBranch3().getRightDataTree();
        }
      else if( (br3x+70 > mx && mx > br3x-3) &&
        (br3y+5 > my && my > br3y-10) ) {
         return finalT.getBranch3().getLeftDataTree();
        }
// .................
      else if( (b4x+70 > mx && mx > b4x-3) &&
        (b4y+5 > my && my > b4y-10) ) {
         return finalT.getBranch4().getDataTree();
        }
      else if( (bl4x+70 > mx && mx > bl4x-3) &&
        (bl4y+5 > my && my > bl4y-10) ) {
         return finalT.getBranch4().getRightDataTree();
        }
      else if( (br4x+70 > mx && mx > br4x-3) &&
        (br4y+5 > my && my > br4y-10) ) {

         return finalT.getBranch4().getLeftDataTree();
        }
// .................
      else if( (b5x+70 > mx && mx > b5x-3) &&
        (b5y+5 > my && my > b5y-10) ) {
         return finalT.getBranch5().getDataTree();
        }
      else if( (bl5x+70 > mx && mx > bl5x-3) &&
        (bl5y+5 > my && my > bl5y-10) ) {
         return finalT.getBranch5().getRightDataTree();
        }
      else if( (br5x+70 > mx && mx > br5x-3) &&
        (br5y+5 > my && my > br5y-10) ) {
         return finalT.getBranch5().getLeftDataTree();
        }
//.................
    else {
        //t.setCurrent(t.getBranch5());
         return finalT.getBranch5().getDataTree();
```

```
        }
}

public BinaryTree currentFinalT(int mx,int my) {

      if( (b6x+70 > mx && mx > b6x-3) &&
          (b6y+5 > my && my > b6y-10) ) {
         finalT.setCurrent(finalT.getBranch6());
          return finalT ;
         }
     else if( (bl6x+70 > mx && mx > bl6x-3) &&
          (bl6y+5 > my && my > bl6y-10) ) {
         finalT.setCurrent(finalT.getBranch6());
          return finalT ;
         }
      else if( (br6x+70 > mx && mx > br6x-3) &&
          (br6y+5 > my && my > br6y-10) ) {
         finalT.setCurrent(finalT.getBranch6());
          return finalT ;
         }
 //..................
     else if( (b7x+70 > mx && mx > b7x-3) &&
          (b7y+5 > my && my > b7y-10) ) {
         finalT.setCurrent(finalT.getBranch7());
          return finalT ;
         }
     else if( (bl7x+70 > mx && mx > bl7x-3) &&
          (bl7y+5 > my && my > bl7y-10) ) {
         finalT.setCurrent(finalT.getBranch7());
          return finalT ;
         }
      else if( (br7x+70 > mx && mx > br7x-3) &&
          (br7y+5 > my && my > br7y-10) ) {
         finalT.setCurrent(finalT.getBranch7());
          return finalT ;
         }
//..................
     else if( (b8x+70 > mx && mx > b8x-3) &&
          (b8y+5 > my && my > b8y-10) ) {
         finalT.setCurrent(finalT.getBranch8());
          return finalT ;
         }
     else if( (bl8x+70 > mx && mx > bl8x-3) &&
          (bl8y+5 > my && my > bl8y-10) ) {
         finalT.setCurrent(finalT.getBranch8());
          return finalT ;
         }
      else if( (br8x+70 > mx && mx > br8x-3) &&
          (br8y+5 > my && my > br8y-10) ) {
         finalT.setCurrent(finalT.getBranch8());
          return finalT ;
         }
 //..................
     else if( (b9x+70 > mx && mx > b9x-3) &&
          (b9y+5 > my && my > b9y-10) ) {
         finalT.setCurrent(finalT.getBranch9());
          return finalT ;
         }
     else if( (bl9x+70 > mx && mx > bl9x-3) &&
          (bl9y+5 > my && my > bl9y-10) ) {
         finalT.setCurrent(finalT.getBranch9());
          return finalT ;
         }
      else if( (br9x+70 > mx && mx > br9x-3) &&
          (br9y+5 > my && my > br9y-10) ) {
         finalT.setCurrent(finalT.getBranch9());
          return finalT ;
```

```
        }
//.................
     else if( (b10x+70 > mx && mx > b10x-3) &&
         (b10y+5 > my && my > b10y-10) ) {
        finalT.setCurrent(finalT.getBranch10());
          return finalT ;
         }
     else if( (bl10x+70 > mx && mx > bl10x-3) &&
         (bl10y+5 > my && my > bl10y-10) ) {
        finalT.setCurrent(finalT.getBranch10());
          return finalT ;
         }
      else if( (br10x+70 > mx && mx > br10x-3) &&
         (br10y+5 > my && my > br10y-10) ) {
        finalT.setCurrent(finalT.getBranch10());
          return finalT ;
         }
// . . . . . .
     else if( (b11x+70 > mx && mx > b11x-3) &&
         (b11y+5 > my && my > b11y-10) ) {
        finalT.setCurrent(finalT.getBranch11());
          return finalT ;
         }
     else if( (bl11x+70 > mx && mx > bl11x-3) &&
         (bl11y+5 > my && my > bl11y-10) ) {
        finalT.setCurrent(finalT.getBranch11());
          return finalT ;
         }
      else if( (br11x+70 > mx && mx > br11x-3) &&
         (br11y+5 > my && my > br11y-10) ) {
        finalT.setCurrent(finalT.getBranch11());
          return finalT ;
         }
// . . . . . .

     if( (b1x+70 > mx && mx > b1x-3) &&
         (b1y+5 > my && my > b1y-10) ) {
        finalT.setCurrent(finalT.getBranch1());
          return finalT ;
         }
     else if( (bl1x+70 > mx && mx > bl1x-3) &&
         (bl1y+5 > my && my > bl1y-10) ) {
        finalT.setCurrent(finalT.getBranch1());
          return finalT ;
         }
      else if( (br1x+70 > mx && mx > br1x-3) &&
         (br1y+5 > my && my > br1y-10) ) {
        finalT.setCurrent(finalT.getBranch1());
          return finalT ;
         }
   //.........................
      else if( (b2x+70 > mx && mx > b2x-3) &&
         (b2y+5 > my && my > b2y-10) ) {
        finalT.setCurrent(finalT.getBranch2());
          return finalT ;
         }
     else if( (bl2x+70 > mx && mx > bl2x-3) &&
         (bl2y+5 > my && my > bl2y-10) ) {
        finalT.setCurrent(finalT.getBranch2());
          return finalT ;
         }
      else if( (br2x+70 > mx && mx > br2x-3) &&
         (br2y+5 > my && my > br2y-10) ) {
        finalT.setCurrent(finalT.getBranch2());
          return finalT ;
         }
  //.........................
```

```
      else if( (b3x+70 > mx && mx > b3x-3) &&
          (b3y+5 > my && my > b3y-10) ) {
         finalT.setCurrent(finalT.getBranch3());
          return finalT ;
         }
      else if( (bl3x+70 > mx && mx > bl3x-3) &&
          (bl3y+5 > my && my > bl3y-10) ) {
         finalT.setCurrent(finalT.getBranch3());
          return finalT ;
         }
       else if( (br3x+70 > mx && mx > br3x-3) &&
          (br3y+5 > my && my > br3y-10) ) {
         finalT.setCurrent(finalT.getBranch3());
          return finalT ;
         }
  //.............................
      else if( (b4x+70 > mx && mx > b4x-3) &&
          (b4y+5 > my && my > b4y-10) ) {
         finalT.setCurrent(finalT.getBranch4());
          return finalT ;
         }
      else if( (bl4x+70 > mx && mx > bl4x-3) &&
          (bl4y+5 > my && my > bl4y-10) ) {
         finalT.setCurrent(finalT.getBranch4());
          return finalT ;
         }
       else if( (br4x+70 > mx && mx > br4x-3) &&
          (br4y+5 > my && my > br4y-10) ) {
         finalT.setCurrent(finalT.getBranch4());
          return finalT ;
         }
  //.............................
      else if( (b5x+70 > mx && mx > b5x-3) &&
          (b5y+5 > my && my > b5y-10) ) {
         finalT.setCurrent(finalT.getBranch5());
          return finalT ;
         }
      else if( (bl5x+70 > mx && mx > bl5x-3) &&
          (bl5y+5 > my && my > bl5y-10) ) {
         finalT.setCurrent(finalT.getBranch5());
          return finalT ;
         }
       else if( (br5x+70 > mx && mx > br5x-3) &&
          (br5y+5 > my && my > br5y-10) ) {
         finalT.setCurrent(finalT.getBranch5());
          return finalT ;
         }
  //.............................

      else {
          //finalT.setCurrent(finalT.getBranch5());
          return finalT ;
         }
}
public String getDirection(int mx,int my) {

        if( (b1x+70 > mx && mx > b1x-3) &&
         (b1y+5 > my && my > b1y-10) ) {

          return finalT.getBranch1().getNodeTree();
         }
        else if( (bl1x+70 > mx && mx > bl1x-3) &&
         (bl1y+5 > my && my > bl1y-10) ) {

          return "left";
         }
         else if( (br1x+70 > mx && mx > br1x-3) &&
```

```
        (br1y+5 > my && my > br1y-10) ) {

         return "right";
        }
  //.........
        else if( (b2x+70 > mx && mx > b2x-3) &&
        (b2y+5 > my && my > b2y-10) ) {

         return finalT.getBranch2().getNodeTree();
        }
      else if( (bl2x+70 > mx && mx > bl2x-3) &&
        (bl2y+5 > my && my > bl2y-10) ) {

         return "left";
        }
        else if( (br2x+70 > mx && mx > br2x-3) &&
        (br2y+5 > my && my > br2y-10) ) {

         return "right";
        }
//...........
        else if( (b3x+70 > mx && mx > b3x-3) &&
        (b3y+5 > my && my > b3y-10) ) {

         return finalT.getBranch3().getNodeTree();
        }
      else if( (bl3x+70 > mx && mx > bl3x-3) &&
        (bl3y+5 > my && my > bl3y-10) ) {

         return "left";
        }
        else if( (br3x+70 > mx && mx > br3x-3) &&
        (br3y+5 > my && my > br3y-10) ) {

         return "right";
        }
//.............
        else if( (b4x+70 > mx && mx > b4x-3) &&
        (b4y+5 > my && my > b4y-10) ) {

         return finalT.getBranch4().getNodeTree();
        }
      else if( (bl4x+70 > mx && mx > bl4x-3) &&
        (bl4y+5 > my && my > bl4y-10) ) {

         return "left";
        }
        else if( (br4x+70 > mx && mx > br4x-3) &&
        (br4y+5 > my && my > br4y-10) ) {

         return "right";
        }
//.............
       else if( (b5x+70 > mx && mx > b5x-3) &&
        (b5y+5 > my && my > b5y-10) ) {

         return finalT.getBranch5().getNodeTree();
        }
      else if( (bl5x+70 > mx && mx > bl5x-3) &&
        (bl5y+5 > my && my > bl5y-10) ) {

         return "left";
        }
        else if( (br5x+70 > mx && mx > br5x-3) &&
        (br5y+5 > my && my > br5y-10) ) {

         return "right";
```

```
        }
  //.............
        else if( (b6x+70 > mx && mx > b6x-3) &&
         (b6y+5 > my && my > b6y-10) ) {

          return finalT.getBranch6().getNodeTree();
         }
       else if( (bl6x+70 > mx && mx > bl6x-3) &&
         (bl6y+5 > my && my > bl6y-10) ) {

          return "left";
         }
         else if( (br6x+70 > mx && mx > br6x-3) &&
         (br6y+5 > my && my > br6y-10) ) {

          return "right";
         }
 //.............
        else if( (b7x+70 > mx && mx > b7x-3) &&
         (b7y+5 > my && my > b7y-10) ) {

          return finalT.getBranch7().getNodeTree();
         }
       else if( (bl7x+70 > mx && mx > bl7x-3) &&
         (bl7y+5 > my && my > bl7y-10) ) {

          return "left";
         }
         else if( (br7x+70 > mx && mx > br7x-3) &&
         (br7y+5 > my && my > br7y-10) ) {

          return "right";
         }
//.............
        else if( (b8x+70 > mx && mx > b8x-3) &&
         (b8y+5 > my && my > b8y-10) ) {

          return finalT.getBranch8().getNodeTree();
         }
       else if( (bl8x+70 > mx && mx > bl8x-3) &&
         (bl8y+5 > my && my > bl8y-10) ) {

          return "left";
         }
         else if( (br8x+70 > mx && mx > br8x-3) &&
         (br8y+5 > my && my > br8y-10) ) {

          return "right";
         }
//.............
        else if( (b9x+70 > mx && mx > b9x-3) &&
         (b9y+5 > my && my > b9y-10) ) {

          return finalT.getBranch9().getNodeTree();
         }
       else if( (bl9x+70 > mx && mx > bl9x-3) &&
         (bl9y+5 > my && my > bl9y-10) ) {

          return "left";
         }
         else if( (br9x+70 > mx && mx > br9x-3) &&
         (br9y+5 > my && my > br9y-10) ) {

          return "right";
         }
 //.............
        else if( (b10x+70 > mx && mx > b10x-3) &&
```

```
          (b10y+5 > my && my > b10y-10) ) {

           return finalT.getBranch10().getNodeTree();
          }
        else if( (bl10x+70 > mx && mx > bl10x-3) &&
          (bl10y+5 > my && my > bl10y-10) ) {

           return "left";
          }
          else if( (br10x+70 > mx && mx > br10x-3) &&
          (br10y+5 > my && my > br10y-10) ) {

           return "right";
          }
//.................
        else if( (b11x+70 > mx && mx > b11x-3) &&
          (b11y+5 > my && my > b11y-10) ) {

           return finalT.getBranch11().getNodeTree();
          }
        else if( (bl11x+70 > mx && mx > bl11x-3) &&
          (bl11y+5 > my && my > bl11y-10) ) {

           return "left";
          }
          else if( (br11x+70 > mx && mx > br11x-3) &&
          (br11y+5 > my && my > br11y-10) ) {

           return "right";
          }
     else {
          //t.setCurrent(t.getBranch5());
           return "error";
          }
}
public boolean mouseDown(Event e,int x,int y)
  {
       mousex = x;
       mousey = y;
       return true;
  }
}// end class
```

APPENDIX B

SEMANTIC TABLEAUX RULES

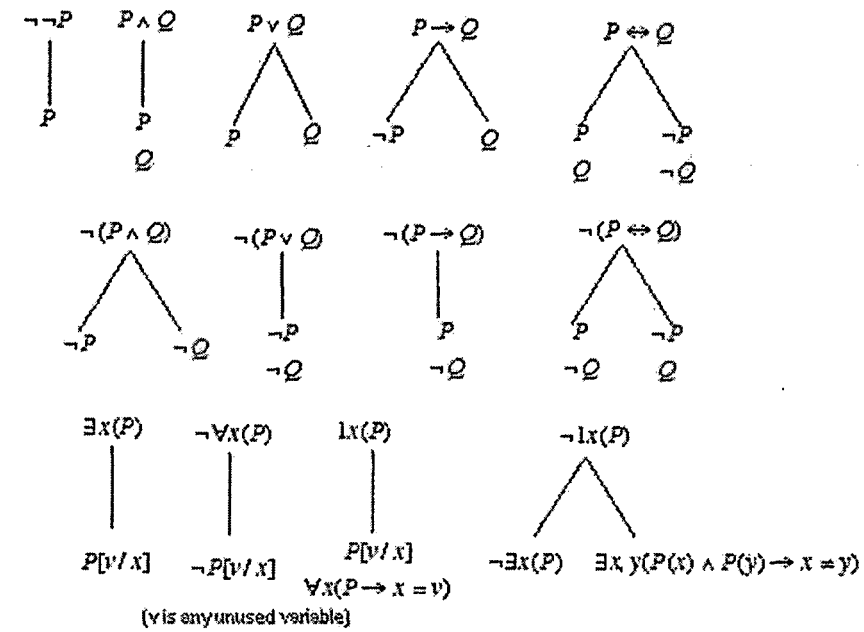
Semantic Tableaux (After W Hodges).

To show $P, Q \cdots \vdash R$, start with $P, Q \cdots, \neg R$

Use any rule of algebra or Boolean algebra to replace formulae.

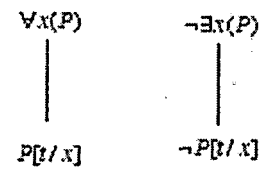
P
|
Q

Apply following rule and check off the formula:



(v is any unused variable)

Apply the following rule but do not check off the source – it may needed again with a different substitution.



(t is any term that is free for x in P)

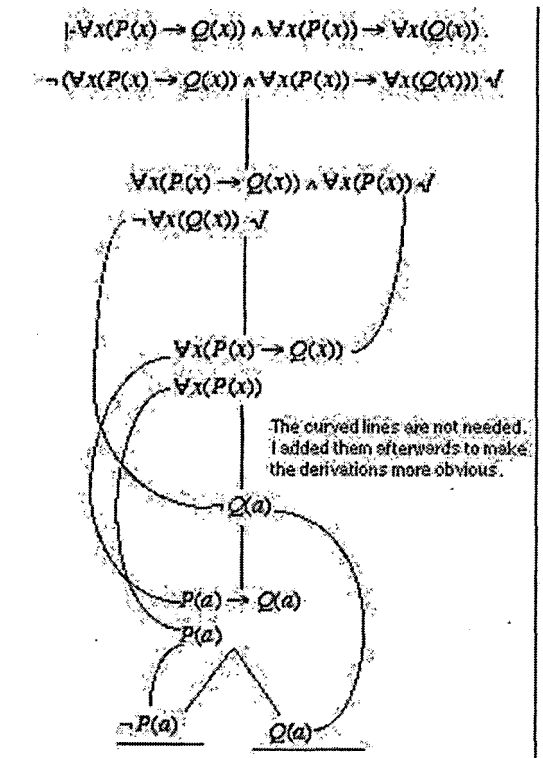
Close any branch that has both P and $\neg P$ or with $t \neq t$

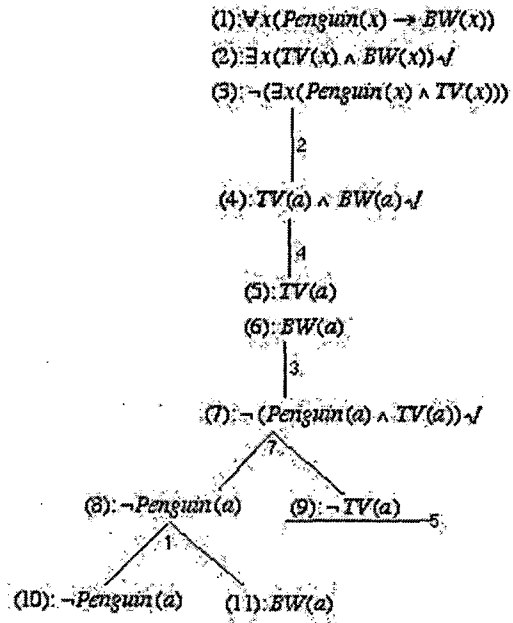
If all branches are closed then the intial set of formulae are inconsistent, alternately any open branch is a possible consequence of assuming the initial formulae.

APPENDIX C

SEMANTIC TABLEAUX EXAMPLES

Example Semantic Tableau



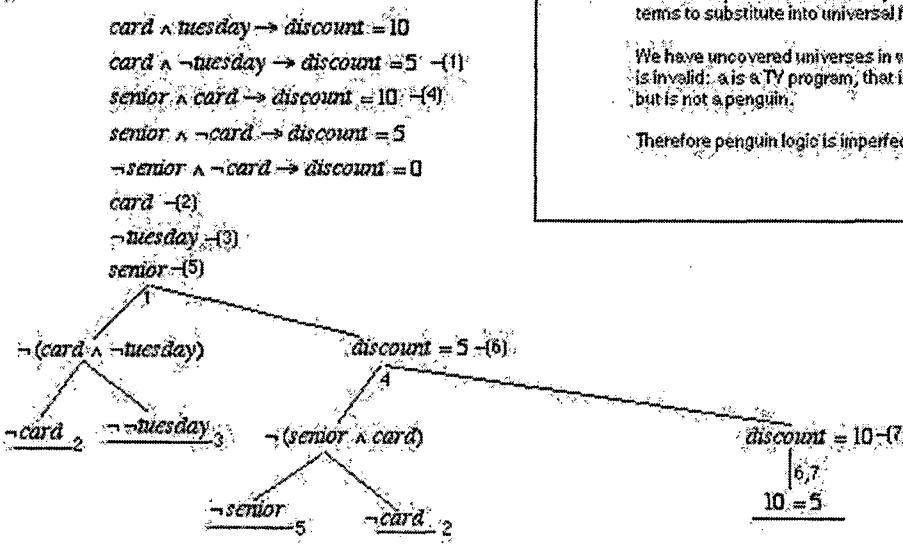


Note: I took the bother to label the formulae. But I did several steps in one go.

This is an open tableau. Further, we have no more terms to substitute into universal forms.

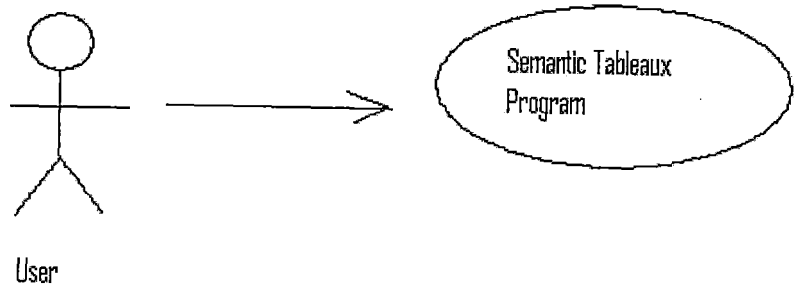
We have uncovered universes in which the argument is invalid: a is a TV program, that is black and white, but is not a penguin.

Therefore penguin logic is imperfect in this case.



APPENDIX D

USE CASE


Semantic Tableaux
Program
User

APPENDIX E

UNIT TEST

Program	Action	Result
ChildNode.java	Data structure to create a node.	Passed or Failed
ChildNode()	Node containing all the necessary information.	Passed
ChildNode(String d)	Set String 'd' to ChildNode's data.	Passed
setLeft(ChilNode l)	Set another ChildNode left of the current one.	Passed
setRight (ChildNode r)	Set another ChildNode right of the current one.	Passed
setParent (ChildNode p)	Set a ChildNode as parent (above) of the current one.	Passed
setData(String d)	Set ChildNode's 'data' to 'd'.	Passed
setLeftData (String d)	Set ChildNode's 'LData' to 'd'.	Passed
setRightData (String d)	Set ChildNode's 'RData' to 'd'.	Passed
setQuantifier (String d)	Set ChildNode's 'quantifier' to 'd'.	Passed
setPredicate (String d)	Set ChildNode's 'predicate' to 'd'.	Passed
setOperator (String d)	Set ChildNode's 'operator' to 'd'.	Passed
setVariable1 (String d)	Set ChildNode's 'variable1' to 'd'.	Passed
setVariable2 (String d)	Set ChildNode's 'variable2' to 'd'.	Passed
setDTestClose (String d)	Set ChildNode's 'DTestClose' to 'd'.	Passed
setRTestClose (String d)	Set ChildNode's 'RTestClose' to 'd'.	Passed
setLTestClose (String d)	Set ChildNode's 'LTestClose' to 'd'.	Passed
setDataTree(BinaryTree d)	Set ChildNode's 'dataTree' to 'd'.	Passed
setRightDataTree (BinaryTree d)	Set ChildNode's 'rightDataTree' to 'd'.	Passed
setLeftDataTree (BinaryTree d)	Set ChildNode's 'leftDataTree' to 'd'.	Passed
setLeftTree (String d)	Set ChildNode's 'leftTree' to 'd'.	Passed
setRightTree (String d)	Set ChildNode's 'rightTree' to 'd'.	Passed
setNodeTree (String d)	Set ChildNode's 'nodeTree' to 'd'.	Passed
setNot()	Switch ChildNode's boolean 'not'.	Passed
setLNot()	Switch ChildNode's	Passed

	boolean 'Lnot'.	
setRNot()	Switch ChildNode's boolean 'Rnot'.	Passed
setCenter()	Switch ChildNode's boolean 'center'.	Passed
setCheckedOff()	Switch ChildNode's boolean 'checkedOff'.	Passed
setRCheckedOff()	Switch ChildNode's boolean 'RcheckedOff'.	Passed
setLCheckedOff()	Switch ChildNode's boolean 'LcheckedOff'.	Passed
setClosed()	Switch ChildNode's boolean 'closed'.	Passed
setRClosed()	Switch ChildNode's boolean 'Rclosed'.	Passed
setLClosed()	Switch ChildNode's boolean 'Lclosed'.	Passed
setLineClosed()	Switch ChildNode's boolean 'lineClosed'.	Passed
setClicked()	Switch ChildNode's boolean 'clicked'.	Passed
setRClicked()	Switch ChildNode's boolean 'Rclicked'.	Passed
setLClicked()	Switch ChildNode's boolean 'Lclicked'.	Passed
getLeft()	Return node to current node's left.	Passed
getRight()	Return node to current node's right.	Passed
getParent()	Return node to current node's parent.	Passed
getNot()	Return the boolean value for ChildNode's 'not'.	Passed
getLnot()	Return the boolean value for ChildNode's 'Lnot'.	Passed
getRnot()	Return the boolean value for ChildNode's 'Rnot'.	Passed
getCenter()	Return the boolean value for ChildNode's 'center'.	Passed
getCheckedOff()	Return the boolean value for ChildNode's 'checkedOff'.	Passed
getRCheckedOff()	Return the boolean value for ChildNode's 'RcheckedOff'.	Passed
getLCheckedOff()	Return the boolean value for ChildNode's 'LcheckedOff'.	Passed
getClosed()	Return the boolean value for ChildNode's 'closed'.	Passed
getRClosed()	Return the boolean value for ChildNode's	Passed

	'Rclosed'.	
getLClosed()	Return the boolean value for ChildNode's 'Lclosed'.	Passed
getLineClosed()	Return the boolean value for ChildNode's 'lineClosed'.	Passed
getClicked()	Return the boolean value for ChildNode's 'clicked'.	Passed
getRClicked()	Return the boolean value for ChildNode's 'Rclicked'.	Passed
getLClicked()	Return the boolean value for ChildNode's 'Lclicked'.	Passed
getData()	Return the String value for ChildNode's 'data'.	Passed
getLeftData()	Return the String value for ChildNode's 'LData'.	Passed
getRightData()	Return the String value for ChildNode's 'RData'.	Passed
getDTestClosed()	Return the String value for ChildNode's 'DTestClosed'.	Passed
getRTestClosed()	Return the String value for ChildNode's 'RTestClosed'.	Passed
getLTestClosed()	Return the String value for ChildNode's 'LTestClosed'.	Passed
getQuantifier()	Return the String value for ChildNode's 'quantifier'.	Passed
getOperator()	Return the String value for ChildNode's 'operator'.	Passed
getPredicate()	Return the String value for ChildNode's 'predicate'.	Passed
getVariable1()	Return the String value for ChildNode's 'variable1'.	Passed
getVariabale2()	Return the String value for ChildNode's 'variable2'.	Passed
getDataTree()	Return the BinaryTree Data structure for ChildNode's 'dataTree'.	Passed
getRightDataTree()	Return the BinaryTree Data structure for ChildNode's	Passed

	'rightDataTree'.	
getLeftDataTree()	Return the BinaryTree Data structure for ChildNode's 'leftDataTree'.	Passed
getLeftTree()	Return the String value for ChildNode's 'leftTree'.	Passed
getRightTree()	Return the String value for ChildNode's 'rightTree'.	Passed
getNodeTree()	Return the String value for ChildNode's 'nodeTree'.	Passed

Program	Action	Result
BinaryTree.java	Data structure to create a BinaryTree.	Passed or Failed
setRoot(ChildNode r)	Set the ChildNode that will be the root. Also set 'current' and branches 1-11 also at root.	Passed
setCurrent (ChildNode n)	Set ChildNode 'n' to the current node.	Passed
BinaryTree()	Set root to null.	Passed
isEmpty()	Return boolean value if root is equal to null.	Passed
getRoot()	Get the 'root' ChildNode.	Passed
getCurrent()	Get the 'current' ChildNode.	Passed
getBranch1()	Get the 'branch1' ChildNode.	Passed
getBranch2()	Get the 'branch2' ChildNode.	Passed
getBranch3()	Get the 'branch3' ChildNode.	Passed
getBranch4()	Get the 'branch4' ChildNode.	Passed
getBranch5()	Get the 'branch5' ChildNode.	Passed
getBranch6()	Get the 'branch6' ChildNode.	Passed
getBranch7()	Get the 'branch7' ChildNode.	Passed
getBranch8()	Get the 'branch8' ChildNode.	Passed
getBranch9()	Get the 'branch9'	Passed

	ChildNode.	
getBranch10()	Get the 'branch10' ChildNode.	Passed
getBranch11()	Get the 'branch11' ChildNode.	Passed
setBranch1 (ChildNode b)	Set ChildNode 'b' as 'branch1'.	Passed
setBranch2 (ChildNode b)	Set ChildNode 'b' as 'branch2'.	Passed
setBranch3 (ChildNode b)	Set ChildNode 'b' as 'branch3'.	Passed
setBranch4 (ChildNode b)	Set ChildNode 'b' as 'branch4'.	Passed
setBranch5 (ChildNode b)	Set ChildNode 'b' as 'branch5'.	Passed
setBranch6 (ChildNode b)	Set ChildNode 'b' as 'branch6'.	Passed
setBranch7 (ChildNode b)	Set ChildNode 'b' as 'branch7'.	Passed
setBranch8 (ChildNode b)	Set ChildNode 'b' as 'branch8'.	Passed
setBranch9 (ChildNode b)	Set ChildNode 'b' as 'branch9'.	Passed
setBranch10 (ChildNode b)	Set ChildNode 'b' as 'branch10'.	Passed
setBranch11 (ChildNode b)	Set ChildNode 'b' as 'branch11'.	Passed
getNot()	Return the boolean value for the current ChildNode's 'not'.	Passed
getLNot()	Return the boolean value for the current ChildNode's 'Lnot'.	Passed
getRNot()	Return the boolean value for the current ChildNode's 'Rnot'.	Passed
getCenter()	Return the boolean value for the current ChildNode's 'center'.	Passed
getCheckedOff()	Return the boolean value for the current ChildNode's 'checkedOff'.	Passed
getClosed()	Return the boolean value for the current ChildNode's 'closed'.	Passed
getData()	Return the String value for current ChildNode's 'data'	Passed
getRightData()	Return the String value for current ChildNode's 'Rdata'.	Passed

getLeftData()	Return the String value for current ChildNode's 'Ldata'.	Passed
getQuantifier()	Return the String value for current ChildNode's 'quantifier'.	Passed
getOperator()	Return the String value for current ChildNode's 'operator'.	Passed
getPredicate()	Return the String value for current ChildNode's 'predicate'	Passed
getVariable1()	Return the String value for current ChildNode's 'variable1'.	Passed
getVariable2()	Return the String value for current ChildNode's 'variable2'.	Passed
getRight()	Get current ChildNode's right ChildNode.	Passed
getLeft()	Get current ChildNode's left ChildNode.	Passed
getDataTree()	Get current ChildNode's 'dataTree'.	Passed
getRightDataTree()	Get current ChildNode's 'rightdataTree' .	Passed
getLeftDataTree()	Get current ChildNode's 'leftdataTree'.	Passed
getLeftTree()	Get Current ChildNode's 'getLeftTree'.	Passed
getRightTree()	Get Current ChildNode's 'getRightTree'.	Passed
getNodeTree()	Get Current ChildNode's 'getNodeTree'.	Passed
getParent()	Get current ChildNode's parent ChildNode.	Passed
setNot()	Set boolean value for current ChildNode's 'not'	Passed

setLNot()	Set boolean value for current ChildNode's 'Lnot'.	Passed
setRNot()	Set boolean value for current ChildNode's 'Rnot'.	Passed
setCenter()	Set boolean value for current ChildNode's 'center'.	Passed
setCheckedOff()	Set boolean value for current ChildNode's 'checkedOff'.	Passed
setClosed()	Set boolean value for current ChildNode's 'closed'.	Passed
inData(String o)	Set String 'o' as current ChildNode's 'data' value.	Passed
inLeftData(Sting o)	Set String 'o' as current ChildNode's 'LData' value.	Passed
inRightData (Sting o)	Set String 'o' as current ChildNode's 'RData' value.	Passed
inQuantifier(Sting o)	Set String 'o' as current ChildNode's 'quantifier' value.	Passed
inOperator(String o)	Set String 'o' as current ChildNode's 'operator' value.	Passed
inPredicate (String o)	Set String 'o' as current ChildNode's 'predicate' value.	Passed
inVariable1 (String o)	Set String 'o' as current ChildNode's 'variable1' value.	Passed
inVariable2 (String o)	Set String 'o' as current ChildNode's 'variable2' value.	Passed
inDataTree (BinaryTree o)	Set BinaryTree 'o' to current ChildNode's 'dataTree' value.	Passed
inRightDataTree (BinaryTree o)	Set BinaryTree 'o' to current ChildNode's 'rightDataTree' value.	Passed
inLeftDataTree (BinaryTree o)	Set BinaryTree 'o' to current ChildNode's 'leftDataTree' value.	Passed
inLeftTree(Sting o)	Set String 'o' as current ChildNode's 'leftTree' value	Passed
inRightTree(Sting o)	Set String 'o' as	Passed

	current ChildNode's 'rightTree' value.	
inNodeTree(Sting o)	Set String 'o' as current ChildNode's 'nodeTree' value.	Passed
insertLeftGo()	Create left ChildNode for the current ChildNode and go left.	Passed
insertLeft()	Create left ChildNode for the current ChildNode.	Passed
insertRightGo()	Create right ChildNode for the current ChildNode and go Right.	Passed
insertRight()	Create right ChildNode for the current ChildNode.	Passed
insertParentGo()	Create parent ChildNode for the current ChildNode and go to Parent.	Passed
insertParent()	Create parent ChildNode for the current ChildNode.	Passed
pretrav(ChildNode p)	Print out the some of the String variables in the current ChildNode. Do a recursion pretrav on the left childNode. And do a recursion pretrav on the right, until it travels the BinaryTree.	Passed

Program	Action	Result
Pa.java	WFF that was parsed into a vector.	Passed or Failed
Pa(String a)	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed

	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h))))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h))))	Passed
Print()	Output vector onscreen.	Passed

Program	Action	Result
CheckParse.java	Takes a WFF that has Unicode in it and translates it into English.	Passed or Failed
CheckParse (String s)	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x)→¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p)∧∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧∀y Q(q)∨(K(l)↔∀y(H(h))))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h))))	Passed

Program	Action	Result
BuildTree.java	Takes a vector of WFF formula and builds a binary tree.	Passed or Failed
BuildTree (Vector c)	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h))))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h))))	Passed

Program	Action	Result
ParseTree. java	Takes a binary tree with a correct WFF formula in it and builds a string.	Passed or Failed
ParseTree (BinaryTree d)	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p)∧∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(1)∧ ∀y Q(q)∨(K(1)↔∀y(H(h)))	Passed
	1x((P(p)∨∀yQ(q))∨(L(1)↔∀y (H(h)))	Passed

Program	Action	Result
CopyTree. java	Copies a binary tree.	Passed or Failed
CopyTree (BinaryTree d,BinaryTree c)	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p)∧∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(1)∧ ∀y Q(q)∨(K(1)↔∀y(H(h)))	Passed
	1x((P(p)∨∀yQ(q))∨(L(1)↔∀y (H(h)))	Passed

Program	Action	Result
ReplaceVariable. java	Replaces a free variable with another variable in the appropriate place in a binary tree.	Passed or Failed
ReplaceVariable (BinaryTree d,String	Replace 'r' with 'x' in ∀x((P(x) ∨Q(y,x)) ∨	Passed

a,String b))	∀y(K(x,y)↔H(x)))	
	Replace 'r' with 'x' in ∀y((P(x,y) ∨Q(x)) ∨ ∀x(K(x)↔H(y,x)))	Passed

Program	Action	Result
ReplaceCapture.java	Replaces a variable with another variable in a capture situation.	Passed or Failed
ReplaceCapture (BinaryTree d, String c,String a)	Replace 'r' with 'x' in ∀x((P(x) ∨Q(x)) ∨ ∀x(K(x)↔H(x)))	Passed
	Replace 'r' with 'x' in ∀x((P(x) ∨Q(x)) ∨ ∀y(K(x)↔H(x)))	Passed

Program	Action	Result
SearchTree.java	Sees if the variable the user selected to replace is in the scope of another quantifier. Returns the variable or returns "ok".	Passed or Failed
SearchTree (BinaryTree c, String d)	∀y((P(p)∨ ∀y Q(q))∨ ∀l(K(l)↔H(h)))	Passed
	∀x((P(p)∨ ∀y Q(q))∨ ∀y(K(l)↔H(h)))	Passed

Program	Action	Result
CaptureTree.java	Returns a variable that is free in a WFF.	Passed or Failed
CaptureTree (BinaryTree d)	∀x((P(p)∨ ∀y Q(q))∨ ∀y(K(l)↔H(h)))	Passed

Program	Action	Result
STree.java(must have working ReplaceVariable.java ,CopyTree.java ,Pa.java ,and ParseTree.java)	Creates a Semantic Tableaux tree by looking at a certain node in a binary tree. The program sees if the node is an operator or quantifier and creates a new branch in the Semantic Tableaux tree and	Passed or Failed

	assigns a branch number to the new branch or branches.	
Stree(BinaryTree d,BinaryTree c,String a,String b)	$\forall$	Passed
	$\exists$	Passed
	1	Passed
	$\neg\forall$	Passed
	$\neg\exists$	Passed
	$\neg$1	Passed
	$\wedge$	Passed
	$\vee$	Passed
	$\rightarrow$	Passed
	$\leftrightarrow$	Passed
	$\neg\wedge$	Passed
	$\neg\vee$	Passed
	$\neg\rightarrow$	Passed
	$\neg\leftrightarrow$	Passed
NewTreeSetRight (BinaryTree c)	Creates a right branch on the Semantic Tableaux tree and assigns a branch number.	Passed
NewTreeSetLeft (BinaryTree c)	Creates a left branch on the Semantic Tableaux tree and assigns a branch number.	Passed
BranchSetLeft (BinaryTree c)	Goes left on a binary tree and assigns a branch number.	Passed
BranchSetRight (BinaryTree c)	Goes right on a binary tree and assigns a branch number.	Passed

Program	Action	Result
Interface.java	Input GUI applet	Passed or Failed
Interface()	Initiates the GUI buttons and text field.	Passed
action(Event event,Object ob)	"Clear" button clears text. "Assumption1" button enters a String as first Assumption. "Assumption2" button enters a String as	Passed

	second Assumption. "Conclusion" button enters a String as Conclusion. "Correct" accepts a String. "Incorrect" clears a String. "For_All" enters "∀" into the text String. "There_Exists" enters "∃" into the text String. "And" enters "∧" into the text String. "Or" enters "∨" into the text String. "Not" enters "¬" into the text String. "If_then" enters "→" into the text String. "If_And_Only_If" enters "↔" into the text String.	
getA1(String a)	Returns entered String.	Passed
getA2(String a)	Returns entered String.	Passed
getCon(String a)	Returns entered String.	Passed

Program	Action	Result
Output.java	Second GUI applet that creates the Semantic Tableaux tree.	Passed or Failed
init()	Initiates GUI buttons and text field. Creates binary tree from Assumption1, Assumption2, and Conclusion Strings.	Passed
action(Event e, Object o) (must have working ParseTree.java, ,CopyTree.java ,SearchTree.java ,ReplaceCapture.java .CaptureTree.java	"One" button starts a new node in a new Semantic Tableaux tree from Assumption1 binary tree. Or "One" gives instruction to use	Passed

ReplaceVariable.java ,CopyTree.java ,Pa.java ,and ParseTree.java)	Assumption1 binary tree to create a new node in an existing Semantic Tableaux tree. "Two" button starts a new node in a new Semantic Tableaux tree from Assumption2 binary tree. Or "Two" gives instruction to use Assumption2 binary tree to create a new node in an existing Semantic Tableaux tree." Three" button starts a new node in a new Semantic Tableaux tree from Conclusion binary tree. Or "Three" gives instruction to use Conclusion binary tree to create a new node in an existing Semantic Tableaux tree." Enter" button starts a new node in a Semantic Tableaux tree by using a binary tree previously selected in "One","Two", or "Three". Or creates a new node in a Semantic Tableaux tree by using the WFF the user selected on an existing Semantic Tableaux tree.	
paint(Graphics g)	Draws the Semantic Tableaux tree in progress.	Passed
moveRight (BinaryTree d)	A test to see if the Semantic Tableaux tree has a right node.	Passed
moveLeft(BinaryTree d)	A test to see if the Semantic Tableaux	Passed

	tree has a left node.	
moveParent (BinaryTree d)	A test to see if the Semantic Tableaux tree has a parent node.	Passed
BinaryTree closedT(BinaryTree d)	Sees if a branch in the Semantic Tableaux tree has a contradiction in it.	Passed
CheckQ(String b,BinaryTree d)	Make sure the user is forming the Semantic Tableaux tree properly.	Passed
checkOne(String b,Binarytree d){	Make sure that the "1" quantifier has a unused variable for substitution.	Passed
whichBranch (BinaryTree d)	Identifies the branch number of a certain node.	Passed
sendBranchBack (BinaryTree d)	Returns the branch number of a node back to the root.	Passed
isCheckedOff (BinaryTree d)	A test to see if the WFF in a node has been checked off.	Passed
Branchxy(String b, int x, int y)	Assign the coordinates to a branch.	Passed
LBranchxy(String b, int x, int y)	Assign the coordinates to the 'left' data of a node.	Passed
RBranchxy(String b, int x, int y)	Assign the coordinates to the 'right' data of a node.	Passed
getWorkBranch (String a, BinarTree d)	Set a branch to the current node.	Passed
doSTree1()	Calculate a new Semantic Tableaux tree from branch 1.	Passed
doSTree2()	Calculate a new Semantic Tableaux tree from branch 2.	Passed
doSTree3()	Calculate a new Semantic Tableaux tree from branch 3.	Passed
doSTree4()	Calculate a new Semantic Tableaux tree from branch 4.	Passed
doSTree5()	Calculate a new	Passed

	Semantic Tableaux tree from branch 5.	
getBranchxy (int x, int y))	From the coordinates gotten from the mouse, get the String stored in the Semantic Tableaux tree.	Passed
getBranch (int x, int y)	From the coordinates gotten from the mouse, the binary tree stored in the selected node is gotten.	Passed
currentFinal (int x, int y)	From the coordinates gotten from the mouse, the branch selected in the Semantic Tableaux tree is set at the current branch.	Passed
getDirection (int x, int y)	From the coordinates gotten from the mouse, the appropriate WFF of a node is selected.	Passed
mouseDown (Event e, int x, int y)	Mouse coordinates are collected.	Passed

Program	Action	Result
Sending.java	Sends the three WFF gotten from Interface.java to Output.java.	Passed or Failed
setA1(String a)	Gets one WFF from Interface.java.	Passed
setA2(String a)	Gets one WFF from Interface.java.	Passed
setC1(String a)	Gets one WFF from Interface.java.	Passed
getA1	Gets the WFF to Output.java.	Passed
getA2	Gets the WFF to Output.java.	Passed
getC1	Gets the WFF to Output.java.	Passed

APPENDIX F

SUBSYSTEM TESTING

Program	Action	Result
Pa.java and BuildTree.java	Parses a WFF string to a vector and creates a binary tree	Passed or Failed
	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h)))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h)))	Passed

Program	Action	Result
BuildTree.java and ParseTree.java	Creates a WFF binary tree and parses it.	Passed or Failed
	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h)))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h)))	Passed

Program	Action	Result
BinaryTree.java and ReplaceVariable.java	Creates a binary tree and replaces a variable in it.	Passed or Failed
	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h)))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h)))	Passed

Program	Action	Result
BinaryTree.java and CopyTree.java	Creates a binary tree and copies it.	Passed or Failed
	P(x)	Passed
	¬P(x)∨Q(x)	Passed
	¬P(x) → ¬Q(x)	Passed
	∀x(P(x)→Q(x))	Passed
	∃x(P(x)→Q(x))	Passed
	∀x ∃y,h 1z(P(p,h))	Passed
	∃x 1y ¬∃z(P(p,x)↔ ¬Q(q,y))	Passed
	∀x(P(p) ∧ ∃x∃d¬(¬G(g)∨D(d)))	Passed
	∀x(L(l)∧ ∀y Q(q)∨(K(l)↔∀y(H(h)))	Passed
	1x((P(p)∨∀y Q(q))∨(L(l)↔∀y(H(h)))	Passed

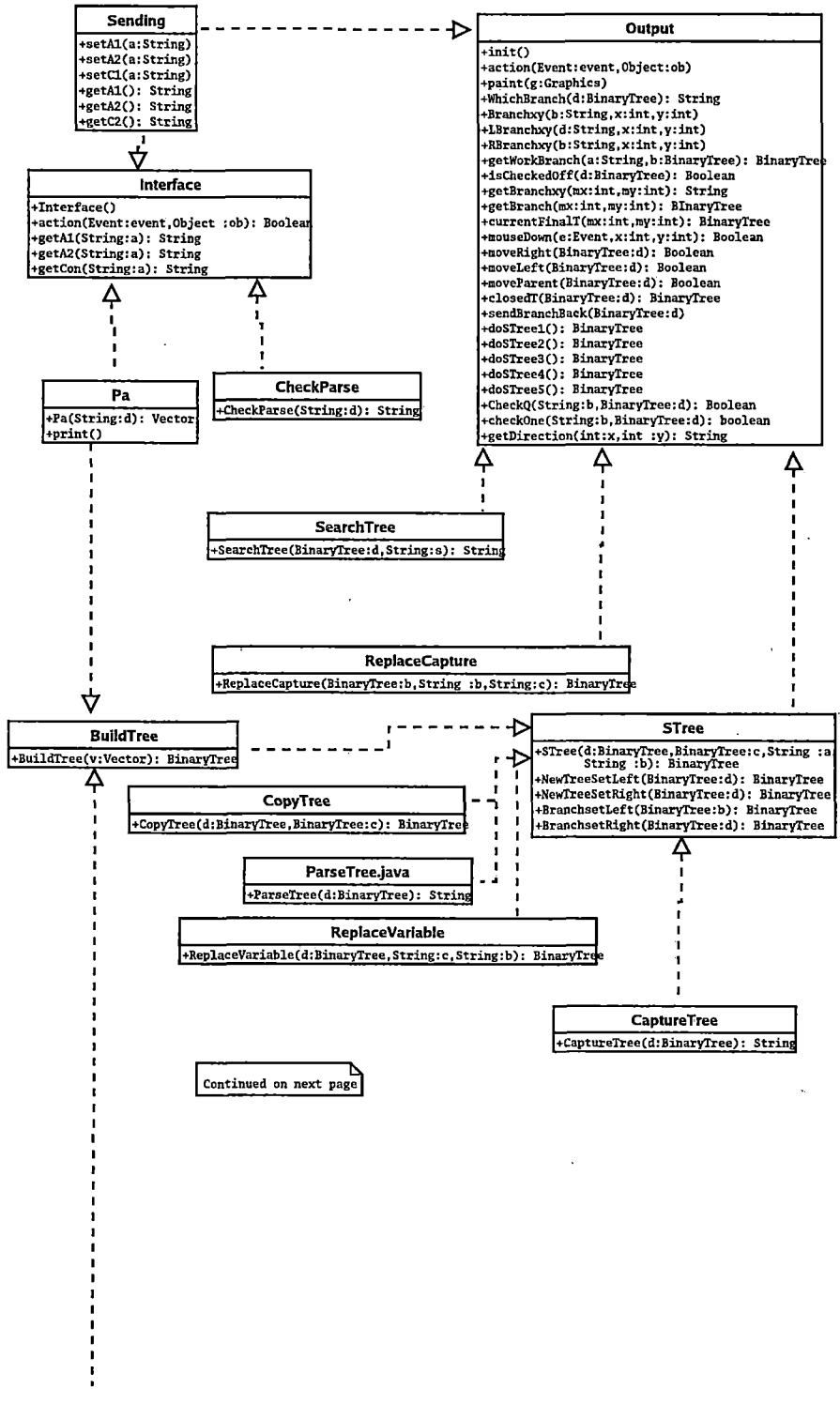
APPENDIX G

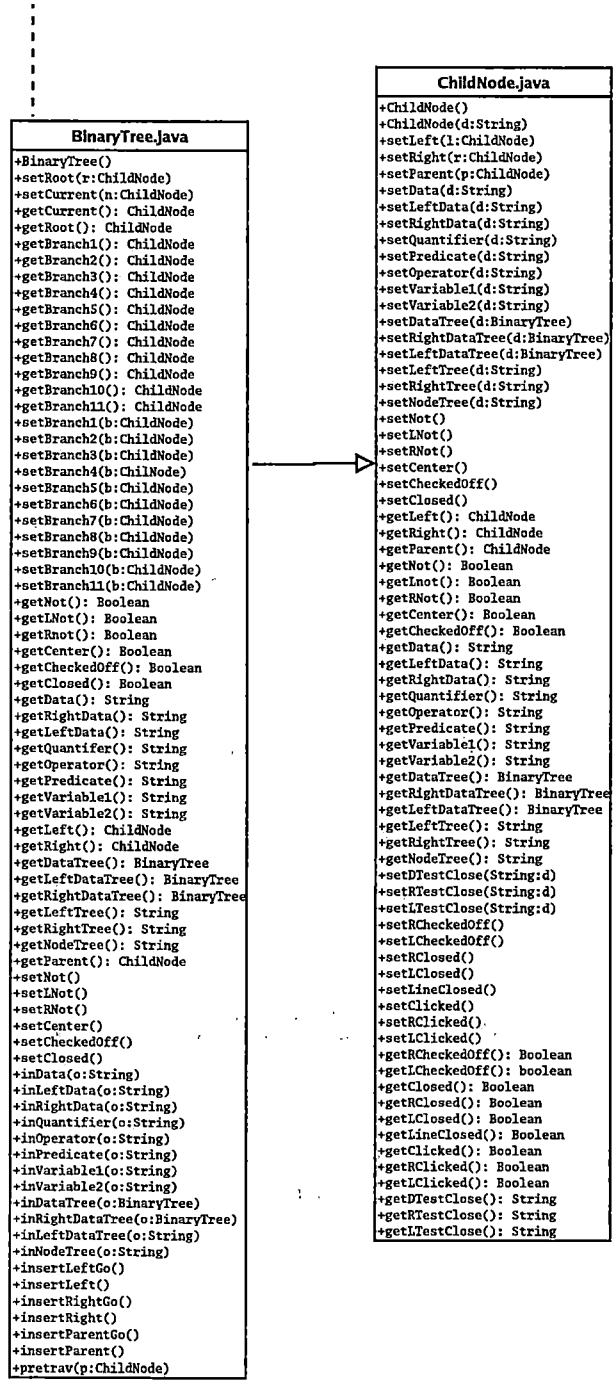
PROJECT TEST

Semantic Tableaux Project	Action	Result
	Does the Interface.html and Output.html loads fast?	Yes
	Can you click to the Output.html?	Yes
	Do you see instructions in the Output.html text box?	Yes
	WFF that has been checked of in grey?	Yes
	Do you see the WFF the user selected in green?	Yes
	Are the contradictions in red? And is there a red line that goes up to where the branch splits?	Yes
	Can you select the WFF branch with ease?	Yes

APPENDIX H

UML CLASS DIAGRAM


Sending
+setA1(a:String)
+setA2(a:String)
+setC1(a:String)
+getA1(): String
+getA2(): String
+getC2(): String
Output
+init()
+action(Event:event,Object:ob)
+paint(g:Graphics)
+WhichBranch(d:BinaryTree): String
+Branchxy(b:String,x:int,y:int)
+LBranchxy(d:String,x:int,y:int)
+RBranchxy(b:String,x:int,y:int)
+getWorkBranch(a:String,b:BinaryTree): BinaryTree
+isCheckedOff(d:BinaryTree): Boolean
+getBranchxy(mx:int,my:int): String
+getBranch(mx:int,my:int): BInaryTree
+currentFinalT(mx:int,my:int): BinaryTree
+mouseDown(e:Event,x:int,y:int): Boolean
+moveRight(BinaryTree:d): Boolean
+moveLeft(BinaryTree:d): Boolean
+moveParent(BinaryTree:d): Boolean
+closedT(BinaryTree:d): BinaryTree
+sendBranchBack(BinaryTree:d)
+doSTree1(): BinaryTree
+doSTree2(): BinaryTree
+doSTree3(): BinaryTree
+doSTree4(): BinaryTree
+doSTree5(): BinaryTree
+CheckQ(String:b,BinaryTree:d): Boolean
+checkOne(String:b,BinaryTree:d): boolean
+getDirection(int:x,int :y): String
Interface
+Interface()
+action(Event:event,Object :ob): Boolean
+getA1(String:a): String
+getA2(String:a): String
+getCon(String:a): String
Pa
+Pa(String:d): Vector
+print()
CheckParse
+CheckParse(String:d): String
SearchTree
+SearchTree(BinaryTree:d,String:s): String
ReplaceCapture
+ReplaceCapture(BinaryTree:b,String :b,String:c): BinaryTree
BuildTree
+BuildTree(v:Vector): BinaryTree
STree
+STree(d:BinaryTree,BinaryTree:c,String :a
String :b): BinaryTree
+NewTreeSetLeft(BinaryTree:d): BinaryTree
+NewTreeSetRight(BinaryTree:d): BinaryTree
+BranchsetLeft(BinaryTree:b): BinaryTree
+BranchsetRight(BinaryTree:d): BinaryTree
CopyTree
+CopyTree(d:BinaryTree,BinaryTree:c): BinaryTree
ParseTree.java
+ParseTree(d:BinaryTree): String
ReplaceVariable
+ReplaceVariable(d:BinaryTree,String:c,String:b): BinaryTree
CaptureTree
+CaptureTree(d:BinaryTree): String
Continued on next page


BinaryTree.Java
+BinaryTree()
+setRoot(r:ChildNode)
+setCurrent(n:ChildNode)
+getCurrent(): ChildNode
+getRoot(): ChildNode
+getBranch1(): ChildNode
+getBranch2(): ChildNode
+getBranch3(): ChildNode
+getBranch4(): ChildNode
+getBranch5(): ChildNode
+getBranch6(): ChildNode
+getBranch7(): ChildNode
+getBranch8(): ChildNode
+getBranch9(): ChildNode
+getBranch10(): ChildNode
+getBranch11(): ChildNode
+setBranch1(b:ChildNode)
+setBranch2(b:ChildNode)
+setBranch3(b:ChildNode)
+setBranch4(b:ChilNode)
+setBranch5(b:ChildNode)
+setBranch6(b:ChildNode)
+setBranch7(b:ChildNode)
+setBranch8(b:ChildNode)
+setBranch9(b:ChildNode)
+setBranch10(b:ChildNode)
+setBranch11(b:ChildNode)
+getNot(): Boolean
+getLNot(): Boolean
+getRnot(): Boolean
+getCenter(): Boolean
+getCheckedOff(): Boolean
+getClosed(): Boolean
+getData(): String
+getRightData(): String
+getLeftData(): String
+getQuantifer(): String
+getOperator(): String
+getPredicate(): String
+getVariable1(): String
+getVariable2(): String
+getLeft(): ChildNode
+getRight(): ChildNode
+getDataTree(): BinaryTree
+getLeftDataTree(): BinaryTree
+getRightDataTree(): BinaryTree
+getLeftTree(): String
+getRightTree(): String
+getNodeTree(): String
+getParent(): ChildNode
+setNot()
+setLNot()
+setRNot()
+setCenter()
+setCheckedOff()
+setClosed()
+inData(o:String)
+inLeftData(o:String)
+inRightData(o:String)
+inQuantifier(o:String)
+inOperator(o:String)
+inPredicate(o:String)
+inVariable1(o:String)
+inVariable2(o:String)
+inDataTree(o:BinaryTree)
+inRightDataTree(o:BinaryTree)
+inLeftDataTree(o:String)
+inNodeTree(o:String)
+insertLeftGo()
+insertLeft()
+insertRightGo()
+insertRight()
+insertParentGo()
+insertParent()
+pretrav(p:ChildNode)
ChildNode.java
+ChildNode()
+ChildNode(d:String)
+setLeft(l:ChildNode)
+setRight(r:ChildNode)
+setParent(p:ChildNode)
+setData(d:String)
+setLeftData(d:String)
+setRightData(d:String)
+setQuantifier(d:String)
+setPredicate(d:String)
+setOperator(d:String)
+setVariable1(d:String)
+setVariable2(d:String)
+setDataTree(d:BinaryTree)
+setRightDataTree(d:BinaryTree)
+setLeftDataTree(d:BinaryTree)
+setLeftTree(d:String)
+setRightTree(d:String)
+setNodeTree(d:String)
+setNot()
+setLNot()
+setRNot()
+setCenter()
+setCheckedOff()
+setClosed()
+getLeft(): ChildNode
+getRight(): ChildNode
+getParent(): ChildNode
+getNot(): Boolean
+getLnot(): Boolean
+getRNot(): Boolean
+getCenter(): Boolean
+getCheckedOff(): Boolean
+getData(): String
+getLeftData(): String
+getRightData(): String
+getQuantifier(): String
+getOperator(): String
+getPredicate(): String
+getVariable1(): String
+getVariable2(): String
+getDataTree(): BinaryTree
+getRightDataTree(): BinaryTree
+getLeftDataTree(): BinaryTree
+getLeftTree(): String
+getRightTree(): String
+getNodeTree(): String
+setDTestClose(String:d)
+setRTestClose(String:d)
+setLTestClose(String:d)
+setRCheckedOff()
+setLCheckedOff()
+setRClosed()
+setLClosed()
+setLineClosed()
+setClicked()
+setRClicked()
+setLClicked()
+getRCheckedOff(): Boolean
+getLCheckedOff(): boolean
+getClosed(): Boolean
+getRClosed(): Boolean
+getLClosed(): Boolean
+getLineClosed(): Boolean
+getClicked(): Boolean
+getRClicked(): Boolean
+getLClicked(): Boolean
+getDTestClose(): String
+getRTestClose(): String
+getLTestClose(): String

REFERENCES

[1] Langley Formal Methods (2003,Sept 26). [online].
http://shemesh/larc.nasa.gov/fm/

[2] Semantic Tableaux Proof Method for Predicate Logic (2004, Jan 16).[online].
http://www.everything2.com/index.pl?node_id=1513374

[3] Twootie and Bertie3 Home Page (2005, Jan).[online].
http://www.ucc.uconn.edu/~wwwphil/software.html

[4] Predicate Logic(1994, August 19).[online].
http://www.cee.hw.ac.uk/~alison/ai3notes/section2_4_3.htm

[5] Reiner Hähnle, "Tableaux and Related Methods" in Handbook of Automated Reasoning Volume 1, Alan Robinson and Andrei Voronkov, Ed. Cambridge:MIT Press,2001,pp. 103-177.

[6]Richard C. Jeffrey, Formal Logic:Its Scope and Limits. McGraw-Hill Book Company, 1967, page 64.

[7] Reiner Hähnle, "Tableaux and Related Methods" in Handbook of Automated Reasoning Volume 1, Alan Robinson and Andrei Voronkov, Ed. Cambridge:MIT Press,2001,pp. 103-177.

[8] Michael R.A. Huth and Mark D. Ryan, Logic in Computer Science, Modelling and reasoning about systems, Cambridge University Press, 2000,page. 107.

[9] Modus Ponens (2002,Oct 6).[online]. http://lc.brooklyn.cuny.edu/LeftBarFiles/FromAboutLC/Core5Files/Logic/ponens.html

[10] The Bertie/Twootie Home Page(2002,June).[online]. http://www.ucc.uconn.edu/~wwwphil/software.html

[11] Twootietwo (2005, September).[online] http://www.philosophy.ubc.ca/faculty/burkholder/220/twootietwo/

[12] Chen-Hsiu(Jimmy)Lee,"A Tabular Propositional Logic: And/Or Table Translator, MS CSCI Project, Spring 2003, California State University, San Bernardino, 92407

[13] Javacommerce(2005, September).[online] http://www.javacommerce.com/tutorial/JavaData/javaData.html

[14] Java World(2005, March).[online] http://www.javaworld.com/javaworld/javatips/jw-javatip3.html